Neural Architecture and Feature Search for Predicting the Ridership of Public Transportation Routes

Afiya Ayman*, Juan Martinez†, Philip Pugliese‡, Abhishek Dubey†, Aron Laszka*
*University of Houston, †Vanderbilt University, ‡Chattanooga Area Regional Transportation Authority

***Abstract*—Accurately predicting the ridership of public-transit routes provides substantial benefits to both transit agencies, who can dispatch additional vehicles proactively before the vehicles that serve a route become crowded, and to passengers, who can avoid crowded vehicles based on publicly available predictions. The spread of the coronavirus disease has further elevated the importance of ridership prediction as crowded vehicles now present not only an inconvenience but also a public-health risk. At the same time, accurately predicting ridership has become more challenging due to evolving ridership patterns, which may make all data except for the most recent records stale. One promising approach for improving prediction accuracy is to fine-tune the hyper-parameters of machine-learning models for each transit route based on the characteristics of the particular route, such as the number of records. However, manually designing a machine-learning model for each route is a labor-intensive process, which may require experts to spend a significant amount of their valuable time. To help experts with designing machine-learning models, we propose a neural-architecture and feature search approach, which optimizes the architecture and features of a deep neural network for predicting the ridership of a public-transit route. Our approach is based on a randomized local hyper-parameter search, which minimizes both prediction error as well as the complexity of the model. We evaluate our approach on real-world ridership data provided by the public transit agency of Chattanooga, TN, and we demonstrate that training neural networks whose architectures and features are optimized for each route provides significantly better performance than training neural networks whose architectures and features are generic.**

I. Introduction

Public transit is an integral part of modern metropolitan cities as it enables diverse groups of people to access services and jobs. Better public transit means less congestion, faster commutes, increased social equity, and overall lower carbon footprint [1, 2]. However, transit services are often severely stressed or underdeveloped in practice. One of the primary reasons for this is the inherent challenge in maximizing ridership while providing fair coverage under resource constraints. Transit authorities try to balance this tension between ridership and scope; however, due to extreme spatial and temporal heterogeneity in demand, the resulting schedules are often not satisfactory to the needs of many demographics. This lack of accessibility to public transit negatively affects the lives of residents in a multitude of ways: it poses barriers to educational attainment, accessing community services, and accessing jobs.

Further, passenger crowding has always been a problem for public transportation since it deteriorates the passengers' well-being and satisfaction. With the emergence of the novel coronavirus disease (COVID-19) pandemic and social distancing regulations, passengers want to avoid crowds while commuting now more than ever. People who can afford private vehicles want to travel by public transport only if it is reliable, fast, crowd-free, and comfortable.

Accurately predicting the *maximum occupancy of each scheduled transit-vehicle trip* is crucial since it enables the transit agency to prevent crowding by dispatching additional vehicles to serve a transit route, if possible, based on these predictions. This flexibility can help decrease the probability of crowded trips, thereby improving the quality of transit services. Additionally, transit agencies can lower their operational costs by reducing the number of vehicle trips if they are expected to have low occupancy. So, to optimize their services, improve resource efficiency, and ensure passenger satisfaction, public transit agencies must accurately predict the ridership demand of transit trips. These predictions can also help commuters with planing their travels to avoid crowds.

Forecasting ridership demand, however, presents significant challenges. While transit agencies and navigation applications tend to focus on providing vehicle arrival time predictions, relatively little effort is put into predicting the passenger occupancy of the vehicles. In a recently added feature, Google Maps tells commuters how busy a bus is; but this depends largely on user feedback, i.e., users need to estimate independently and provide inputs to indicate how crowded the bus is [3]. This information might be sufficient for everyday commuters in systems with one-way travel patterns. However, different travel patterns would make it difficult for users to determine how crowded the bus will be for other trips on the same route [4]. Further, the majority of studies focus on stop-level ridership [5, 6, 7], but very few consider route-level prediction [8, 9]. There is a substantial difference between the bus-stop and transit-route levels. The maximum occupancy of a vehicle on a transit trip is assessed on the route level. In contrast, all the boardings and alightings at a specific stop are considered on a bus-stop level, and the direction aspect is ignored.

Additionally, since the beginning of the COVID-19 pandemic, there have been drastic changes in ridership patterns worldwide. In prior work, Wilbur et al. demonstrated signifi-

cant differences between the ridership patterns of public transit before and after lockdowns imposed due to the pandemic [10]. Because of the pandemic and associated state restrictions, many public transit agencies introduced fare-free operations and limited the capacity of vehicles to 50% to ensure social distancing and minimize contact, which resulted in a drastic drop in revenues. So, transit agencies significantly lowered vehicle trips to keep costs under control. These changes adopted by the public transport agencies have exacerbated the challenges of the already difficult occupancy-prediction problem. First, datasets that can be used for training have shrunken in size since older data cannot be used due to the changing ridership patterns. So, we have to work with relatively small datasets. Second, the data has a very high variance since ridership demand is more unpredictable than ever.

In this paper, we propose to improve prediction accuracy by *fine-tuning occupancy-prediction models for each transit route*. Manually designing machine-learning models for each route-direction combination is practically intractable since it would require significant time and effort from machine-learning experts. Therefore, we introduce a *neural-architecture and feature search approach that fine-tunes the architecture's hyper-parameters and predictor variables of a deep neural network.* The hyper-parameter and feature-set optimization is based on a randomized local search, which minimizes both prediction error and the complexity of the model. Minimizing the complexity of the model is crucial since it reduces computational cost and time during both training and inference. We define *forecasting the maximum occupancy of transit-vehicle trips on a particular route-direction combination as a specific task* in the context of this paper. The research questions we seek to answer are the following.

Q1 Do task-specific architectures and feature sets, which are fine-tuned for specific transit routes, perform significantly better than generic ones?
Q2 How much impact does the starting architecture of the randomized local search have on the end results?
Q3 How well does the optimized architecture of one task perform with respect to training for other tasks?
Q4 Are there any obvious relations between the characteristics of a task and the complexity of the optimized architecture?

We demonstrate our approach and answer these research questions using automatic passenger count (APC) data from the public transit agency of Chattanooga, TN.

Organization: The remainder of this paper is organized as follows. In Section II, we provide an overview of the state of the art in ridership prediction and neural architecture search. In Section III, we introduce our assumptions about the available ridership data and how we process it. In Section IV, we formulate the architecture and feature search problem and describe our local-search algorithm. In Section V, we present numerical results based on our experiments with real-world data from Chattanooga, TN. Finally, in Section VI, we provide concluding remarks.

II. Related Work

In this section, we provide an overview of the state of the art in predicting the transit ridership (Section II-A) and neural architecture search (Section II-B).

A. Occupancy Prediction

Recent advancements in information and communication technologies have led to several services for transport systems. Mapping the occupancy level of public transport has always held more significance in improving public transit's overall operations. Over recent years, research studies have been trying to map the occupancy level of public transport.

A large stream of literature discusses different aspects of transit ridership. Some approaches investigate the effect of different events, weather, etc., on ridership behavior. Karnberger and Antoniou provide an insight into the relationship between ridership and events in predicting the public transit ridership [9]. Zhou et.al. explore the influence of daily weather condition changes on the usage of public transit [11]. There are several studies where machine learning approaches were adopted for occupancy prediction. Vandewiele et al. proposed classifying the occupancy level of a train [12]. They used time, weather features, and a matrix explaining the connections between origin and destination as training features. First, they train a neural network and later used the output as a feature in the extreme Gradient Boosting (XGBoost) model. Zhang et al. and Silva et al. treated the problem as a regression task [13, 14]. Their experiments show that the XGBoost algorithm outperforms other prominent algorithms on both tasks and reported an RMSE of 25 [13]. There is a group of studies that attempts to predict passenger occupancy on public transport in the near future, using real-time information from smart cards [15, 5, 16]. Tsai et al. presented a study based on a statistical analysis of historical data and compare multiple temporal units NN (MTUNN) and parallel ensemble NN (PENN) with conventional multi-layer perceptron (MLP) [16]. Both MTUNN and PENN outperform MLP. Nuzzolo et al. proposed Short Term Occupancy Prediction (STOP), which predicts the number of passengers on a bus in the nearby future [5]. They designed STOP using Behavioral prediction models, such as ARIMA and RBF neural networks.

While the discussed research attempts provide many interesting insights, there are some fundamental differences with our research. First, we aim to predict the maximum occupancy for the bus on a trip on a given route and heading in a given direction for a certain point in time and location. Second, we have used both sequential and non-sequential features as our predictor variables. We have used a neural network consisting of feedforward and recurrent layers to predict occupancy.

B. Neural Architecture Search

Neural Architecture Search is a technique that aims to outperform the performance of a hand design network by automating the design of an artificial neural network. Although model architectures designed by human experts are often successful, there is no way to be sure whether a better model

exists or not. A systematic and automatic way of learning high-performance model architectures can help find better models.

NAS is very computationally expensive since it explores the search space to propose better architectures and partially or fully train them. Researchers have studied different methods to improve NAS over the past few years. Zoph and Le proposed a reinforcement learning search method with a search space that contained convolutional networks and an RNN based controller to optimize architecture configurations where all the proposed architectures are trained from scratch until convergence [17]. But this approach required high computational power. So, in a later study, Zoph et al. narrowed down the search space and trained the RNN based controller on a proxy dataset. Later on, the learned architecture is transferred to the actual dataset [18]. Pham et al. also use an RNN based controller where the search samples subnetworks from a large parent network [19]. The parameters are shared among the proposed subnetworks instead of trashing them every single time and training from scratch. This approach is less computationally expensive than the previous ones. Along with RL based architecture search, other techniques such as evolutionary algorithm [20], boosting [21], hill-climbing [22], random search [23], etc. have also been used in architecture search. But all the approaches have been evaluated on image data sets.

Our research is different than the existing research attempts for a few reasons. First, despite the popularity of Neural Architecture over the recent years, it has only been used in solving the image classification problems. In our approach, we implement NAS in solving a regression problem, i.e., occupancy prediction . Second, our search space is fundamentally different than the discussed approaches since it consists of both the predictor variables, i.e., input features for the model, and the hyper-parameters of the architectures. Also, the architectures' hyper-parameters include two different types of layers (feedforward and recurrent) in the search space. This work's main contribution is finding the optimal neural network architecture and the set of features that will produce the lowest loss for a particular regression task with the help of Neural Architecture Search.

III. Data Collection and Processing

We first provide an overview of the data sources that we use in our study (Section III-A) and then describe the data processing methods (Section III-B).

A. Data Sources

1) Automatic Passenger Count (APC): Automatic passenger counting systems record the number of people entering and exiting vehicles at different bus stops. Transit authorities install a people counter sensor over each door of the vehicles and these sensors detect the passengers as they enter and exit. Each record in the APC dataset represents an event at a bus stop for one trip servicing on a particular route and a particular direction. For trips on a specific route-direction combination, we aim to predict the maximum number of people on the bus.

2) Weather: Our weather data includes data from multiple weather stations. Based on the geometric location of a particular stop, we identify the nearest weather station and combine the trip's passenger counts with weather information collected at that particular station. The weather features used in our study include temperature, precipitation intensity, humidity, etc.

B. Data Processing

The direction a vehicle is heading towards can affect the level of crowdedness. For example, there can be a lot of boarding events at a particular station in the morning. Still, the crowdedness of different vehicles can differ based on the different directions from the stop. So, each trip in our processed dataset inherently includes a direction. We process the time series APC data recorded from the vehicles by segmenting them in route-direction combination and integrating them with weather, based on location and time. We also calculate previous ridership information for different trips on the same route-direction from historic data and include them with the dataset's existing attributes.

Each trip in the dataset is represented with a fixed-dimension feature space consisting of information about trip, time and weather and also some sequential features which presents previous trips' passenger boarding information and time difference. Before training and testing, we map categorical variables (e.g., day of week) into sets of binary features using one-hot encoding. Our final predictor variables include number of total stops in the trip, month, day of week, time of day, multiple weather attributes as the fixed dimensional features and maximum and median occupancy of n preceding trips with their time-difference as the sequential inputs. The maximum occupancy for a trip on a particular route and direction is the target feature.

The data processing strategies and our proposed method (Section IV) can be applied to any typical dataset from other transit agencies.

IV. Neural Architecture and Feature Search for Occupancy Prediction

Our primary goal in this work is to provide accurate ridership predictions of different public transit routes and help transit agencies ensure better services. To achieve this, we design a machine learning approach and propose an architecture and feature search framework. We want to find an architecture and set of features, $\mathcal{A}^{best}$ that minimize the prediction error and model complexity. We can express our objective as -

$$min_{\mathcal{A}} \ (\ell_{RMSE} + model\ complexity) \tag{1}$$

The prediction error is measured with ***RMSE*** and model complexity is defined in terms of the total number of trainable parameters in the model.

Neural architecture search algorithm consists of three main parts; a search space, a search technique, and a performance evaluation strategy. In this section, we explain our proposed NAS algorithm in detail and explain how the random search

algorithm proposes better architecture with time to minimize the prediction error. Algorithm 1 presents the proposed neural architecture search.

A. Search Space

NAS aims to find optimal training hyper-parameters on a specific architecture search space, including the list of hyper-parameters, to pick from while proposing a new architecture. Since our goal is to optimize the architecture and input features that minimize prediction error and model complexity, our search space consists of both architecture's hyper-parameters and the features.

Our proposed machine learning approach has three modules; a recurrent module, a feed-forward module, and another feed-forward module that combines the output of the two modules. There is a separate input layer to the recurrent module for processing the sequential features and another to the feed-forward module for processing the non-sequential features. All the modules have one or more fully connected hidden layers. The outputs of the feed-forward and recurrent modules are concatenated before going into the final set of feed-forward layers connected to the output neuron. The activation functions for the feed-forward and recurrent module are *Sigmoid* and *Relu* accordingly. The model is optimized using *Adam* optimizer and trained to minimize the *root mean squared error (RMSE)*. Figure 1 shows an example diagram of the prediction model with all the modules. i_F and i_R indicates non-sequential and sequential input features respectively. h_F, h_R, and $h_{F_{comb}}$ indicate the hidden units in feed-forward, recurrent, and combined module and h_{out} indicates the output neuron.

As the occupancy prediction model consists of feed-forward and recurrent neurons and we also aim to get the optimal predictor variables, our search space, Ω consists of multiple hyper-parameters, $\mathcal{HP}$ for both the architecture and input features. Ω includes the number of layers, $\mathcal{L}$ in different modules and the number of neurons, $\mathcal{N}$ in each layer of the modules, and the learning rate α for the model and all non-sequential and sequential features, $\mathcal{F}$.

B. Search Technique

For the search technique, we have used a random localized search algorithm. At each iteration of the NAS, the random search technique will either increase/decrease particular features, $\mathcal{F}$ from the training samples or pick any other hyper-parameter, h to tune with a given probability, $\mathcal{P}_{\mathcal{HP}}$.

When the feature, $\mathcal{F}$ is picked for a particular iteration, it either adds or removes specific attributes from the non-sequential features or increases or decreases the number of sequential inputs with a probability, $\mathcal{P}_{\mathcal{F}}$. The search algorithm tunes the final non-sequential input features by adding or removing different elements from the existing predictor variables at random. It also adjusts the sequential input features by increasing or decreasing the preceding number of trips at random.

For the remaining hyper-parameters of the architecture, such as ($\mathcal{L}$, $\mathcal{N}$ or α), the algorithm is less likely to modify $\mathcal{L}$ in any module. Since adding or removing a hidden layer can be a big step in the search space, higher probabilities are given to changing $\mathcal{N}$ and α. When the algorithm modifies $\mathcal{L}$ in a particular module, it can either add a new layer at the end or remove a hidden layer from the existing ones. $\mathcal{N}$ in the new layer is chosen randomly within one and twice the average of the existing neurons in different layers. When the algorithm picks to modify $\mathcal{N}$ in any particular module, it will randomly choose an $\mathcal{L}$ from the existing layers and modify $\mathcal{N}$ in that layer with a percentage, $\beta_{\mathcal{N}}$. The algorithm can also modify the learning rate, α with a change percentage, β_{α}.

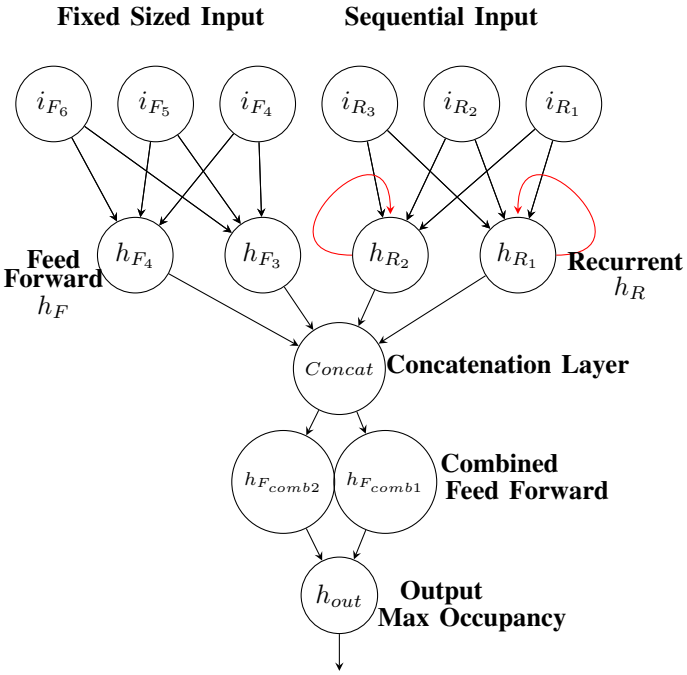


Fig. 1: Example architecture with feed-forward, recurrent, and combined feed-forward modules.

C. Performance Evaluation

Our random search algorithm considers both the loss on the validation set and the model complexity to evaluate a particular architecture. The final *RMSE* is the mean of all *RMSE*s obtained after evaluating the architecture with k-fold cross-validation. Recurrent models trained in practice do not always satisfy stability [24]. Based on our experiments, we have also seen that the performance of the recurrent layers can sometimes be inconsistent. While it happened only a handful of times, we wanted to remove such unstable results from our assessment. So, we evaluate all the architectures using k-fold cross-validation. Again, training and assessing multiple architectures can be computationally costly. So, to minimize time by reducing expensive training, we start with five different 90% / 10% data splits and get five *RMSE*s. Then we check for variations in the observed losses. A decrease in variation should lead to fairer predictability. We use the coefficient of variation, CV, to measure variation. $CV = \sigma/\mu$, where σ is

Algorithm 1: NAS($\mathcal{A}^{start}, t$)

Input : $\mathcal{A} \leftarrow$ start architecture
$t \leftarrow task$
Initialization: $\Omega \leftarrow \{\mathcal{HP}\}$, Search Space
$\mathcal{P}_{\mathcal{HP}} \leftarrow \{$f($\mathcal{HP}$):f($\mathcal{HP}$)$\geq 0, \forall_{\mathcal{HP}} \in \Omega\}$
$\mathcal{A} \leftarrow \mathcal{A}^{start}$
$\beta_\alpha \leftarrow$ change percentage of α
$\beta_\mathcal{N} \leftarrow$ change percentage of $\mathcal{N}$

Function `evaluateArch(`$\mathcal{A}$`)`**:**
  $model \leftarrow$ `trainModel` $(t, \mathcal{A})$
  return $\ell_{RMSE}(t, model) + \omega \cdot model\_complexity$
Function `searchArch(`$\mathcal{A}$`)`**:**
  $\mathcal{HP} = random.choice(\Omega, 1, p = \mathcal{P}_{\mathcal{HP}})$
  if $\mathcal{F}$ *in* $\mathcal{HP}$ **then**
    $\mathcal{A}' \leftarrow$ `changeFeatures(`$\mathcal{HP}$,$\mathcal{A}$`)`
  else if $\mathcal{L}$ *in* $\mathcal{HP}$ **then**
    $\mathcal{A}' \leftarrow$ `changeLayers(`$\mathcal{HP}$,$\mathcal{A}$`)`
  else if $\mathcal{N}$ *in* $\mathcal{HP}$ **then**
    $h_{layer} :=$ random.choice($[0, |\mathcal{A}[\mathcal{HP}]|]$)
    $\mathcal{N}_{curr} \leftarrow \mathcal{A}[\mathcal{HP}][h_{layer}]$
    $\mathcal{A}'[\mathcal{HP}][h_{layer}] \leftarrow \mathcal{N}_{curr} + \lceil \mathcal{N}_{curr} \cdot \beta_\mathcal{N} \rceil$
  else
    $\mathcal{A}'[\alpha] \leftarrow \mathcal{A}[\alpha] + \mathcal{A}[\alpha] \cdot \beta_\alpha$
  return $\mathcal{A}'$
$\mathcal{S}_{prev} =$ `evaluateArch(`$\mathcal{A}$`)`
while $iter < iter^{max}$ **do**
  $\mathcal{A}' \leftarrow$ `searchArch(`$\mathcal{A}$`)`
  $\mathcal{S}' \leftarrow$ `evaluateArch(`$\mathcal{A}'$`)`
  $AcceptProbability \leftarrow exp\left((\mathcal{S}_{prev} - \mathcal{S}') \cdot K\right)$
  $accept \leftarrow random([0, 1])$
  if $accept < AcceptProbability$ **then**
    $\mathcal{A} \leftarrow \mathcal{A}'$
    $\mathcal{S}_{prev} \leftarrow \mathcal{S}'$
  $iter \leftarrow iter + 1$

Result: $\mathcal{A}^{best}$: best architecture found

the standard deviation and μ is the mean. We calculate the CV for errors in each fold, and if the CV is higher than a threshold, we pick 3 new folds to train. We continue this process until the CV is minimized or all the 90% / 10% data splits have been used. From all the prediction errors observed, we remove the couple of best and worst results and average over remaining *RMSE*s obtained this way.

The total number of trainable parameters indicates the model complexity. We consider the model complexity in evaluating performance so that the algorithm proposes smaller networks that will produce a lower loss. The number of trainable parameters of the prediction model is calculated using the connection between layers and biases in each layer [25]. The total number of trainable parameters of the prediction model can be determined with the help of Equation (2).

$$\begin{aligned}\text{train\_params} \rightarrow= i_F \cdot h_{F_0} + h_{F_0} + \sum_{k=1}^{x}(h_{F_{k-1}} \cdot h_{F_k} + h_{F_k}) \\ + h_{R_0} \cdot (i_R + h_{R_0}) + h_{R_0} + \sum_{k=1}^{y}(h_{R_k} \cdot h_{R_{k-1}} + h_{R_k}) \\ + \sum_{k=1}^{n}((h_{F_x} + h_{R_y}) \cdot h_{F_{comb_k}} + h_{F_{comb_k}} \cdot h_{out}) \\ + h_{F_{comb_n}} + h_{out}\end{aligned} \tag{2}$$

Given an architecture set G sampled from Ω and a task, t, our random search algorithm aims to minimize the score $\mathcal{S}$:

$$\mathcal{S} = \ell_{RMSE}(t, model) + \omega \cdot \text{train\_params} \tag{3}$$

where $\ell_{RMSE}(t, model)$ is the RMSE obtained after training and cross validating the model on task, t. Factor ω is a cost ration between prediction error and model complexity.

First, Algorithm 1 obtains an initial score $\mathcal{S}_{prev}$ using the start architecture, $\mathcal{A}^{start}$. Then, it follows an iterative process. In each iteration, the algorithm gets a new architecture, $\mathcal{A}'$ using random localized search of the current best architecture, $\mathcal{A}$ and a new score $\mathcal{S}'$. If the new score, $\mathcal{S}'$ (see Equation (3)) of $\mathcal{A}'$ is lower than the score, $\mathcal{S}_{prev}$ of $\mathcal{A}$, then the algorithm always accepts $\mathcal{A}'$ and $\mathcal{S}'$ as the new solution. Otherwise, the algorithm computes the probability of accepting it based on simulated annealing approach using K and the score difference between $\mathcal{S}_{prev}$ and $\mathcal{S}'$, and then accepts $\mathcal{A}'$ and $\mathcal{S}'$ at random. The algorithm terminates after a fixed number of iterations $iter^{max}$ and returns the best solution, $\mathcal{A}^{best}$ found up to that point.

V. Experimental Setup and Results

In this section, we evaluate our algorithms based on real-world transit data from Chattanooga, TN. We first describe our experimental setup (Section V-A) and then present numerical results (Section V-B). We will publish our dataset and implementation under open-source licenses.

A. Experimental Setup

1) Data: We use APC data for Chattanooga, TN, acquired from the Chattanooga Area Regional Transportation Authority (CARTA). Our study uses 34 months of APC data, from January 2019 to October 2021. CARTA operates a number of transit routes, with trips running in two directions: inbound and outbound. The processed APC dataset from CARTA has trips from 23 routes and both directions. For our study, we have selected 5 routes in both directions, so we have in total 10 different route-direction combinations (i.e., tasks) over which we evaluate our algorithms. We chose these routes to ensure a diverse selection of tasks, considering the number of trips, average occupancy, variance in occupancy values, etc.

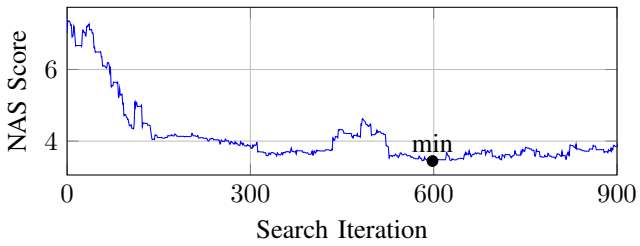


Fig. 2: Neural architecture search with score based on model complexity and the average RMSE over all tasks.

2) Architecture and Feature Search: For the initial architecture of Algorithm 1, we use a neural network that has two hidden layers in each module (feedforward, recurrent, and combined) and a learning rate of 2.5×10^{-5}. The hyper-parameter probability $\mathcal{P}_{\mathcal{HP}}$ is set so that 25% of the time, Algorithm 1 fine-tunes the set of features, and the remaining 75% of the time, it adjusts the hyper-parameters of the architecture search space. For the feature set, we prepare a set of 6 different non-sequential features and the occupancy of 10 preceding trips as the sequential features. Since time information is crucial for predicting the ridership of any route, we always include time features (time of day and day of week) as part of the non-sequential inputs. When the algorithm tunes the feature space, it can pick either sequential or non-sequential features with a uniform probability $\mathcal{P}_{\mathcal{F}}$. The neuron change percentage, $\beta_{\mathcal{N}}$ is chosen uniformly at random from $[-25\%, 25\%]$ in each step. The learning rate change percentage, β_{α} is either -20% or 20%. The NAS algorithm runs for 6,000 iterations before termination, and each architecture $\mathcal{A}$ in G is trained until the learning converges. The training time of each architecture varies based on model complexity and dataset size. Running Algorithm 1 for 6,000 iterations on different datasets takes around 4 to 5 days on average on 1 CPU node with 20 cores.

B. Numerical Results

1) Task-specific vs. Generally Optimized Architecture (Q1): Our first research question is whether the architecture found by a route-direction specific search outperforms a generally optimized architecture. To answer this question, we first run a generic architecture and feature search to find an optimal architecture and feature set that work best on average for all tasks, i.e., over all route-direction combinations. We start the search with all available features and a hand-designed initial architecture. The search algorithm proposes an architecture, and for each task, trains a separate model to obtain an RMSE score. For each task, we use 5–fold cross–validation to obtain an average RMSE score. Finally, the architecture and feature set are accepted or rejected by the search based on model complexity and the average RMSE score over all the tasks. This search returns an architecture and a feature set $\mathcal{A}^{best}$ that work best on average over all tasks. Later, we also use $\mathcal{A}^{best}$ as the starting architecture for the task-specific searches.

Figure 2 shows the first 900 scores of the accepted architectures of the generic search. After trying around 600 different architectures, the search reaches the best architecture based on average *RMSEs* on the validation sets and model complexity.

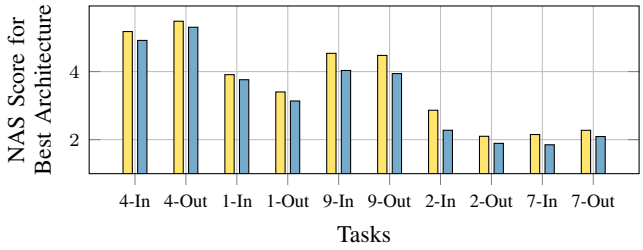


Fig. 3: Comparison between architectures that were found by generic (yellow) and task-specific searches (blue) based on NAS score for each specific task.

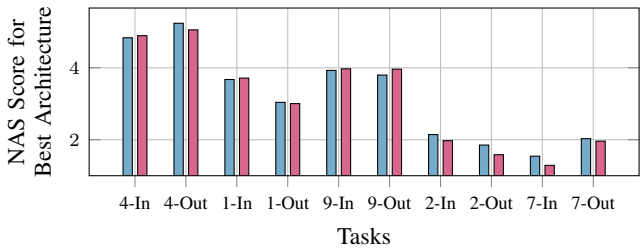


Fig. 4: Comparison between NAS scores attained for each specific task by searches from the hand-designed architecture (blue) and from the best generic architecture (purple).

For the comparison, we run task-specific architecture and feature searches, where we optimize each architectures for predicting the ridership of a specific route and direction. The starting point of each task-specific search is the same hand-designed initial architecture. We run the search for 6000 iterations and select the best architecture. Finally, we retrain the best architecture found by the generic search (Figure 2) on each task, and compare the results with architectures found by task-specific searches.

Figure 3 shows the score after training a specific task with both generic and task-specific architectures. There is a clear gap between the two losses, which indicates that the architecture and feature set found by the task-specific search outperforms the ones found when optimizing for all tasks. So, prediction error and model complexity can be reduced by optimizing the hyper-parameters for a specific task.

2) Starting Architecture of Task-Specific Search (Q2): Our second research question is how much the choice of the initial architecture impacts the performance of the architecture search. To answer this question, we execute the task-specific neural architecture search from two starting points: a hand-designed architecture and the best generic architecture $\mathcal{A}^{best}$ (Figure 2). Each architecture is evaluated based on model complexity and validation set RMSE after 10-fold cross-validation on the specific task.

Table II shows for each task, the NAS score of the best architecture and the time taken by the search to find this architecture from two different starting points (i.e., from hand-

TABLE I: NAS Scores for Models Trained for Various Tasks using Architectures Optimized for Different Tasks

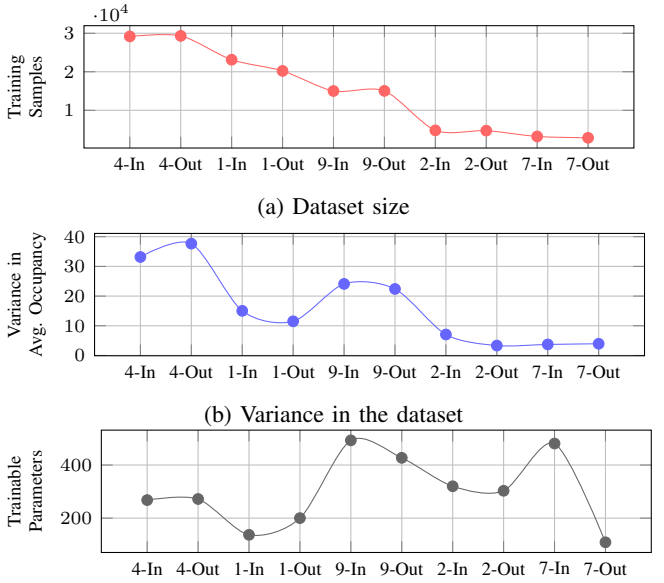
Optimized Arch. \ Task	4 Inbound	4 Outbound	1 Inbound	1 Outbound	9 Inbound	9 Outbound	2 Inbound	2 Outbound	7 Inbound	7 Outbound
4 Inbound	**4.96**	5.30	3.85	3.14	4.66	4.62	2.66	1.94	1.89	2.17
4 Outbound	4.91	**5.09**	3.98	3.14	4.39	4.33	2.68	1.95	1.89	2.19
1 Inbound	5.69	5.81	**3.79**	3.41	4.88	4.48	2.77	2.03	2.07	2.85
1 Outbound	4.94	5.24	3.91	**3.03**	4.37	4.58	2.66	1.94	1.88	2.14
9 Inbound	5.04	5.26	3.95	3.56	**4.52**	4.50	2.75	2.08	1.97	2.37
9 Outbound	5.12	5.42	3.97	3.42	4.64	**4.50**	2.58	2.03	1.96	2.37
2 Inbound	5.06	5.36	3.95	3.18	4.47	4.19	**2.34**	1.57	1.72	2.23
2 Outbound	4.96	5.27	3.81	3.07	4.28	4.21	2.41	**1.72**	1.56	2.15
7 Inbound	5.06	5.26	3.90	3.15	4.17	4.10	2.20	1.69	**1.54**	2.25
7 Outbound	5.58	5.91	3.81	3.38	4.71	4.73	2.64	1.91	1.93	**2.07**
Generic NAS	5.17	5.58	4.02	3.37	4.56	4.61	2.89	2.10	2.17	2.32

TABLE II: Searches with Different Starting Architectures

Task	NAS from Hand-Designed Architecture		NAS from Optimized Architecture	
	Score	Runtime [%]	Score	Runtime [%]
4-In	4.836	47.12	4.894	31.27
4-Out	5.239	65.48	5.056	31.53
1-In	3.675	36.00	3.714	18.57
1-Out	3.039	97.13	3.005	45.88
9-In	3.929	80.15	3.973	87.30
9-Out	3.799	73.23	3.963	86.63
2-In	2.144	61.25	1.973	96.67
2-Out	1.852	65.43	1.584	58.85
7-In	1.543	98.28	1.288	97.63
7-Out	2.030	61.93	1.961	99.03
Average	3.21	68.6	3.14	65.3

designed architecture and from the best generic architecture ($\mathcal{A}^{best}$). Runtime is represented as a percentage of the total number of search iterations. Figure 4 compares the two search results visually based on NAS score. For most tasks (i.e., route-direction combinations), we see that the best architecture found when we initialize the search with the already optimized architecture (found in Section V-B1) performs slightly better. This suggests that we may obtain better NAS results by performing a generic search first, considering all tasks, and then task-specific searches starting from the output of the first search. Further, if we consider the runtime from Table II, we see that the search can converge to the best architecture in slightly less time for most tasks when we start the search from an already optimized architecture.

3) Comparison among Architectures Optimized for Specific Tasks (Q3): Our goal is to assess how an architecture that was optimized for one specific task performs when we use it to train models for other tasks. To this end, we take the best architectures found by the task-specific searches, and we use each architecture to train models for every task in our dataset. Table I shows the NAS scores for all the models based on a 10-fold cross-validation. Each row represents the architecture optimized for a particular task, and each column represents training a model using that architecture for one particular tasks. Darker shades of red and green indicate worse and better scores, respectively. Note that cells on the diagonal show scores for models trained for tasks using their corresponding optimized architectures. We see that for every task, the architecture optimized for that particular task almost always performs best; however, due to the randomness of the search, there are some variations. The last row of Table I shows the NAS score when each task is trained using the best architecture found by the generic search (Figure 2). We can see a significant performance drop in almost every case.



(a) Dataset size

(b) Variance in the dataset

(c) Model complexity of the best architecture

Fig. 5: Characteristics of the tasks and their optimized architectures.

4) Relationship between Characteristics of Tasks and Optimized Architectures (Q4): Figure 5 shows the characteristics of each task in terms of dataset size and variance of occupancy values as well as the complexity of the architecture found by the task-specific search. We observe no obvious relationship between the task and its optimized architecture; we leave a more in depth study of possible relationships to future work.

VI. Conclusion

Accurate prediction of transit ridership provides significant benefits by enabling transit agencies to prevent crowding and passenger to better plan their travel. Due to the challenging nature of this problem, we propose to improve prediction accuracy by fine-tuning machine-learning architectures for each transit route in each direction—a task which could require

significant effort and time from machine-learning experts. Our key contribution is proposing a framework for neural-architecture and feature-set search, which alleviates the need for fine-tuning by machine-learning experts, and demonstrating that our algorithms can significantly reduce prediction error and model complexity based on real-world data. Further, we found that performing a generic search to bootstrap the task-specific searches may slightly reduce runtime; the fact that searches converge to similar architectures regardless of the starting architecture also shows the robustness of our approach.

ACKNOWLEDGEMENTS

This material is based upon work sponsored by the National Science Foundation under grants CNS-1952011, CNS-2029950, and CNS-2029952 and by the Department of Energy under Award Number DE-EE0009212. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or the Department of Energy.

REFERENCES

[1] T. Davis and M. Hale, "Public transportation's contribution to US greenhouse gas reduction," Science Applications International Corporation, Tech. Rep., September 2007.

[2] M. S. P. F. Inambao, "Transportation, pollution and the environment," *International Journal of Applied Engineering Research*, vol. 13, no. 6, pp. 3187–3199, 2018.

[3] Andrew J. Hawkins, "Google will now tell you how crowded your bus or train is likely to be," https://www.theverge.com/2019/6/27/18761187/google-maps-transit-crowded-delays-predictions-train-bus-subway, Accessed: March 8, 2022.

[4] V. V. Gayah, Z. Yu, J. S. Wood, M. N. T. R. Consortium *et al.*, "Estimating uncertainty of bus arrival times and passenger occupancies," Mineta National Transit Research Consortium, Tech. Rep., 2016.

[5] A. Nuzzolo, U. Crisalli, L. Rosati, and A. Ibeas, "STOP: a short term transit occupancy prediction tool for APTIS and real time transit management systems," in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*. IEEE, 2013, pp. 1894–1899.

[6] I. Ceapa, C. Smith, and L. Capra, "Avoiding the crowds: understanding tube station congestion patterns from trip data," in *Proceedings of the ACM SIGKDD international workshop on urban computing*, 2012, pp. 134–141.

[7] S. Bhattacharya, S. Phithakkitnukoon, P. Nurmi, A. Klami, M. Veloso, and C. Bento, "Gaussian process-based predictive modeling for bus ridership," in *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing*, 2013, pp. 1189–1198.

[8] L. Li, J. Wang, Z. Song, Z. Dong, and B. Wu, "Analysing the impact of weather on bus ridership using smart card data," *IET Intelligent Transport Systems*, vol. 9, no. 2, pp. 221–229, 2015.

[9] S. Karnberger and C. Antoniou, "Network–wide prediction of public transportation ridership using spatio–temporal link–level information," *Journal of Transport Geography*, vol. 82, p. 102549, 2020.

[10] M. Wilbur, A. Ayman, A. Ouyang, V. Poon, R. Kabir, A. Vadali, P. Pugliese, D. Freudberg, A. Laszka, and A. Dubey, "Impact of COVID-19 on public transit accessibility and ridership," *arXiv preprint arXiv:2008.02413*, 2020.

[11] M. Zhou, D. Wang, Q. Li, Y. Yue, W. Tu, and R. Cao, "Impacts of weather on public transport ridership: Results from mining data from different sources," *Transportation Research Part C: Emerging Technologies*, vol. 75, pp. 17–29, 2017.

[12] G. Vandewiele, P. Colpaert, O. Janssens, J. Van Herwegen, R. Verborgh, E. Mannens, F. Ongenae, and F. De Turck, "Predicting train occupancies based on query logs and external data sources," in *Proceedings of the 26th International Conference on World Wide Web (WWW)*, 2017, pp. 1469–1474.

[13] N. Zhang, H. Chen, X. Chen, and J. Chen, "Forecasting public transit use by crowdsensing and semantic trajectory mining: Case studies," *ISPRS International Journal of Geo-Information*, vol. 5, no. 10, p. 180, 2016.

[14] R. Silva, S. M. Kang, and E. M. Airoldi, "Predicting traffic volumes and estimating the effects of shocks in massive transportation systems," *Proceedings of the National Academy of Sciences*, vol. 112, no. 18, pp. 5643–5648, 2015.

[15] N. Van Oort, T. Brands, and E. de Romph, "Short term ridership prediction in public transport by processing smart card data," *Transportation Research Record*, 2015.

[16] T.-H. Tsai, C.-K. Lee, and C.-H. Wei, "Neural network based temporal feature models for short-term railway passenger demand forecasting," *Expert Systems with Applications*, vol. 36, no. 2, pp. 3728–3736, 2009.

[17] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[18] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 8697–8710.

[19] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," *arXiv preprint arXiv:1802.03268*, 2018.

[20] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, vol. 33, 2019, pp. 4780–4789.

[21] C. Cortes, X. Gonzalvo, V. Kuznetsov, M. Mohri, and S. Yang, "Adanet: Adaptive structural learning of artificial neural networks," in *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017, pp. 874–883.

[22] T. Elsken, J.-H. Metzen, and F. Hutter, "Simple and

efficient architecture search for convolutional neural networks," *arXiv preprint arXiv:1711.04528*, 2017.

[23] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization." *Journal of Machine Learning Research*, vol. 13, no. 2, 2012.

[24] J. Miller and M. Hardt, "Stable recurrent models," *arXiv preprint arXiv:1805.10369*, 2018.

[25] Raimi Karim, "Counting no. of parameters in deep learning models by hand," https://towardsdatascience.com/counting-no-of-parameters-in-deep-learning-models-by-hand-8f1716241889#192e/, Accessed: Feb 23rd, 2022.