IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 9, 2018, accepted November 13, 2018, date of current version December 31, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2884913

DeepMEC: Mobile Edge Caching Using Deep Learning

KYI THAR[1], NGUYEN H. TRAN[2], (Senior Member, IEEE), THANT ZIN OO[1], AND CHOONG SEON HONG[1], (Senior Member, IEEE)

[1]Department of Computer Science and Engineering, Kyung Hee University, Yongin 17104, South Korea
[2]School of Computer Science, The University of Sydney, Sydney, NSW 2006, Australia

Corresponding author: Choong Seon Hong (cshong@khu.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korean Government (MSIT) under Grant NRF-2017R1A2A2A05000995 and in part by the Institute for Information and Communications Technology Promotion (IITP) Grant funded by the Korean Government (MSIT) through the Development of Access Technology Agnostic Next-Generation Networking Technology for Wired-Wireless Converged Networks under Grant 2015-0-00567.

ABSTRACT Caching popular contents at edge nodes such as base stations is a crucial solution for improving users' quality of services in next-generation networks. However, it is very challenging to correctly predict the future popularity of contents and decide which contents should be stored in the base station cache. Recently, with the advances in big data and high computing power, deep learning models have achieved high prediction accuracy. Hence, in this paper, deep learning is used to learn and predict the future popularity of contents to support cache decision. First, deep learning models are trained and utilized in the cloud data center to make an efficient cache decision. Then, the final cache decision is sent to each base station to store the popular contents proactively. The proposed caching scheme involves three distinct parts: 1) predicting the future class label of each content; 2) predicting the future popularity score of contents based on the predicted class label; and 3) caching the predicted contents with high popularity scores. The prediction models using the Keras and Tensorflow libraries are implemented in this paper. Finally, the performance of the caching schemes is tested with a Python-based simulator. In terms of a cache hit, simulation results show that the proposed scheme outperforms 38%, convolutional recurrent neural network-based scheme outperforms 33%, and convolutional neural network-based scheme outperforms 25% compared to the baseline scheme.

INDEX TERMS Mobile edge computing, deep learning, proactive caching, prediction model searching.

I. INTRODUCTION

According to the Cisco Visual Networking Index, watching videos from wireless handheld devices is generating most of the Internet traffic, and this trend is expected to grow exponentially [1]. To handle this overwhelming Internet traffic, several future Internet network architectures (such as Information Centric Networking and Multi-access Edge Computing) have been proposed with caching capabilities [2]. Thus, utilizing caching capabilities at the edge network reduces the video content's access delay or the number of videos retrievals from the original content server by storing popular videos, where the cache capacity of the edge network is limited. Content popularity can be defined as the ratio of the number of requests for a particular content to the total number of requests, usually obtained for a certain region during a given period of time. The main challenge to designing an efficient caching policy is predicting the future popularity of video contents beforehand, since the video's popularity varies both temporally and spatially [3].

Therefore, we usually assume the popularity of a given content follows some probability distribution, such as a Zipf distribution. Using this assumption, many researchers have proposed various edge-network caching models and cache decision algorithms[1] [4]–[7]. In edge-network caching, cache decision can be classified into two categories: i) reactive caching and ii) proactive caching. In *reactive caching*, the edge node makes the cache decision (whether to store the requested content or not) only when a request for a particular content arrives [8]. In *proactive caching*, the edge node proactively predicts a content's popularity before any user requests are received and makes the cache decision based

[1]Cache decision algorithms are algorithms to make a decisions about whether to store the new content or remove the cached content.

on this [9], [10]. Hence, the cache hit ratio will be affected by the accuracy of the popularity prediction models.

A. RELATED WORK

In practice, the assumption that the content's popularity follows some probability distribution may be invalid because the popularity of the content dynamically changes depending on different factors (e.g, events, type of content, and lifespan of the content). Thus, the predicting popularity of video content has been extensively studied in the literature [9]–[17], but few works consider the integration of forecasting popularity into caching. Many various predictions processes are based on time series models such as autoregressive integrated moving average [18] and classification models [19]. Furthermore, deep learning has achieved huge success in speech recognition, computer vision, and natural language processing. Due to its high accuracy in prediction, deep learning-based prediction models are suitable for content popularity prediction to obtain more accurate results [9]–[13]. Among the various types of the deep learning models, Stack Auto Encoder is used in [9] to predict the popularity of contents in a software-defined network environment. The use of echo state networks, a type of recurrent neural network, in caching is presented in [10]. The use of Deep Neural Networks, an artificial neural network with several hidden layers, in edge network caching is presented in [11]. Collaborative filtering with recurrent neural networks is applied in [12] to predict the popularity of contents. The use of the convolutional neural network in context-aware caching is presented in [13], where the authors utilized social network information to make cache decisions. The main issue of applying deep learning to content popularity prediction is that it is difficult to construct the most suitable models due to the huge amount of hyperparameters configurations. Therefore, in this paper, we present a step-by-step solution for constructing the suitable deep learning models for content popularity prediction.

Previously, deep learning models have not been widely used due to the lack of training data and computing resources. However, deep learning can now be realized because of improvements in computation capacity and the existence of big data to train deep learning models [20]. Big data is usually characterized by the three V's models; volume, variety, and velocity [21], which refer to the large scale of data, different types of data, and speed of the streaming data, respectively. Also, the future Internet must provide ultra-reliable, low-latency communication and intelligently managed devices (such as routers, base stations, and small cell base stations) in real time. Therefore, deep learning (combined with a big data platform) is expected to play an important role in future Internet network architectures that need to learn complex information and enhance the overall network operation. This perspective of utilizing learning models and big data in proactive caching is presented in [22] and [23]. Moreover, [24] and [25] demonstrated neural networks in web proxy cache replacement, and [26] proposed web access pattern prediction by applying neural networks.

To train deep learning models [27], we utilize three types of learning process: i) supervised learning, ii) unsupervised learning, and iii) reinforcement learning. In supervised learning, we know the correct answer (labeled training data) before training our model, which allows us to make predictions about unseen data. In unsupervised learning, we are dealing with unlabeled data or data of an unknown structure. Unsupervised learning techniques extract meaningful information without the guidance of a known outcome variable. In reinforcement learning, we have a sequential decision-making problem, where making a decision influences what decisions can be made in the future. A reward function is provided and it tells us how ''good'' certain states are.

B. CHALLENGES

Deep learning provides prediction results with high accuracy but has several challenging issues for implementation, which are listed as follows:

1) It requires high computation resources to process the big data (high-dimensional data) to train the prediction models.
2) It is difficult to find a suitable prediction model among the various types of deep learning models, such as Multilayer Perceptrons (MLPs), Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Convolutional Recurrent Neural Networks (CRNNs), etc. [27].
3) It is difficult to tune parameters such as the number of layers (i.e., the depth of the network), types of layers (e.g., Convolutional, Recurrent and Fully Connected layers), and learning rate to improve the accuracy of the prediction model.

C. OUR METHODOLOGY

The main goal of this paper is to minimize the user access delay of video content by storing popular video contents at Base Stations (BSs) with the help of a deep learning-based content popularity prediction scheme. We use a supervised learning approach to train the proposed prediction model, which consists of two parts[2]: i) predicting the future popularity class labels of video contents, and ii) predicting the future request counts of video contents. Then, video contents are cached based on predicted results.

Our contributions are summarized as follows:

- An optimization problem is formulated to minimize the content access delay in the near future, i.e., at $t + 1$, by controlling the cache decision at the current time, t, in order to improve the performance of the network.
- To solve the optimization problem at time t with limited information, a deep neural network-based prediction system is proposed to jointly work with the Information Centric Networking enabled mobile edge-computing architecture.

[2]Note that we can directly predict the future request counts of video contents, but the prediction accuracy is low.

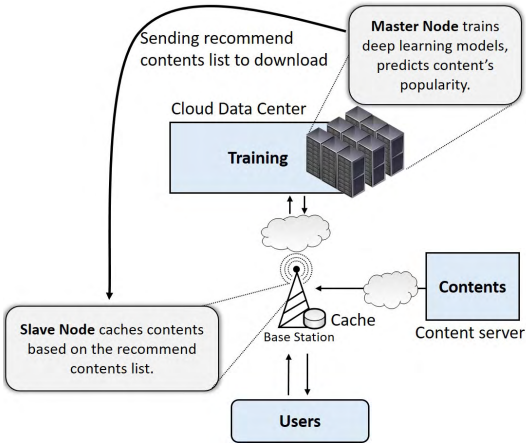

FIGURE 1. System model of learning-based caching at the edge.

- The Movie Lens dataset [28] is analyzed, and based on these findings, possible prediction models are designed, especially considering popularity class prediction and future request count prediction.
- Model generation and a randomized search algorithm are proposed to select the best models suited for future popularity class predictions and future request count predictions, and compared to various types of deep learning models such as CNN, RNN, and CRNN.
- The proposed scheme is trained with the Movie Lens dataset [28] using the machine learning libraries, Keras [29] and Tensorflow [30]. Finally, the performance of the proposed caching scheme is measured using a python-based simulator.

The rest of the paper is organized as follows; The system model is discussed in Sec. II and the problem formulation is presented in Sec. III. Then, the detailed process of predicting the expected request count is presented in Sec. IV. Next, the cache decision algorithm is described in Sec. V. Then, the proposed scheme is evaluated in Sec. VI. Finally, conclusion and future work is presented in Sec. VII.

II. SYSTEM MODEL

The proposed system model is shown in Fig. 1, in which the central controller (*Master Node*) is implemented at the high-end computing node (e.g., cloud data center) and the Slave Node is implemented at the BS. The *Master Node* is responsible for training the deep learning model by utilizing collected data from the BS, then predicts the future popularity of contents with trained models and sends the contents list to the BS to cache popular contents. The *Slave Node* is responsible for storing contents recommended by the *Master Node* and data collection (which collects information such as request counts and the number of cache hits at the BS). If the requested content is located at the BS's cache, the BS immediately replies it to the users. Otherwise, the requested content is retrieved from the content server and then provided to the users.

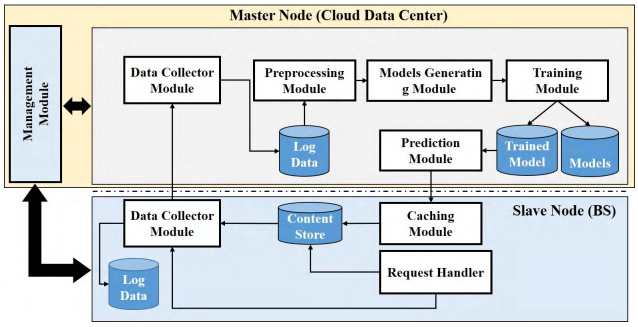

FIGURE 2. An overview system design for learning-based caching.

A. SYSTEM DESIGN OF LEARNING-BASED CACHING

As shown in Fig. 2, the *Master Node* consists of six main components: i) the Management Module, ii) Data Collector Module, iii) Preprocessing Module, iv) Model Generating Module, v) Training Module, and vi) Prediction Module. The Management Module directly controls all of the components of the *Master Node* and *Slave Node*. The Data Collector Module is responsible for collecting data such as request counts for each content, movie ID, etc. The Preprocessing Module is responsible for cleaning, extracting the log files, and constructing the dataset to train the prediction model. Additionally, the Model Generating Module generates various types of prediction models such as CNN and RNN, which are discussed in details in Sec. VI-B. Subsequently, the Training Module handles the training process of the generated models using with a small dataset, and then stores related training information such as the model configuration, training accuracy, and validation accuracy. Then, the Management Module selects the best model, and supervises the training process of the prediction model, including when to stop training and when to store the state of the trained model by controlling the Training Module. Additionally, the Management Module generates and transfers the content lists, which include content IDs, predicted popularity class labels, and predicted request counts to the BS by using the Prediction Module.

The *Slave Node* consists of three main components i) Data Collector Module, ii) Caching Module, and iii) Request Handler Module. The Data Collector collects information such as the number of cache hits and the total number of requests. The Caching module makes the cache decision and downloads contents from a server based on the content list provided by the *Master Node*. The Request Handler Module handles the arriving requests from the users. When request r_k arrives at the node, the Request Handler Module checks the whether requested content is located in its cache or not. If the requested content is in the cache, then the Request Handler Module provides the content to the user. Otherwise, the Request Handler Module downloads the content directly from the servers.

B. MATHEMATICAL MODEL OF LEARNING BASED CACHING

In this section, we define the mathematical model for the caching and prediction processes to solve the content access delay minimization problem as follows. Let the set of contents be denoted by $\mathcal{F} = \{1, \ldots, F\}$. The size of each content f is denoted as κ_f and each BS has a cache capacity S. The number of requests for contents which arrives sequentially at BS can be denoted as $\mathcal{R} = \{r_1, r_2, \ldots, r_k\}$, $k = 1, 2, \ldots$, where r_k is the k^{th} arriving request. Each request r_k includes the following information: $(n_k, \boldsymbol{c}_k, t_k), \forall k$, where n_k is the name of the requested content, t_k is the time stamp of request arrival at the BS, and $\boldsymbol{c}_k$ is the context vector of the k^{th} request. Context $\boldsymbol{c}_k \in \mathcal{R}^d$ is a d-dimensional vector, which includes features such as the movie name, type of the movie and the properties of the content. The BS first stores the arriving requests' information on its local database and then sends this information to the cloud datacenter periodically (e.g., every 6 hours, every night). Then, the *Master Node* constructs the dataset for collected data at total time T and denoted as $\{d^t | t = 1, 2, ..., T\}$, where t is time when the data is collected. Each d^t has the form of a tuple $(\boldsymbol{y}_j^t, \pi_j^t, \phi_j^t) \in \mathbb{R}^J$, where j is the index of the data point number and J is the total number of data points, such as the number of movies.

Inputs features of the data point j for training and prediction is denoted as $\boldsymbol{y}_j \in \mathbb{R}^d$, and which include features such as the movie name or movie ID, the properties of the content. π_j is the true label for the class label prediction, and ϕ_j is the true label for request count prediction. The set of class prediction models is denoted $\mathcal{M}_{\text{class}} = \{1, 2, \ldots, M\}$. The set of request count prediction models is denoted $\mathcal{M}_{\text{req}} = \{1, 2, \ldots, M\}$, where two sets $\mathcal{M}_{\text{class}}$ and $\mathcal{M}_{\text{req}}$ include the same numbers of models M. Each model m belongs to $\mathcal{M}_{\text{class}}$, and $\mathcal{M}_{\text{req}}$ consists of various types of prediction models such as CNN, and RNN. Among them, we choose the best-suited models $\boldsymbol{m}^* = \{(m_{\text{class}}, m_{\text{req}}) | m_{\text{class}}^* \in \mathcal{M}_{\text{class}}, m_{\text{req}}^* \in \mathcal{M}_{\text{req}}\}$, which have the highest validation accuracy for each prediction.

III. PROBLEM FORMULATION

In this section, we formulate the learning-based caching scheme as an access delay minimization problem for a single BS. For each arriving request r_k from the user, the BS first checks whether the requested content is located in its cache or not. We assume that if the requested content is located in the BS's cache, the BS provides the contents to the user with no delay. Otherwise, the BS provides the content to the user with a delay δ per megabyte of content size.

A. CACHING MODEL AND CONSTRAINTS

Let us denote the cache size of the BS as s, in gigabytes. We define a decision variable for caching content at the BS at time t as $x_f^t \in \{0, 1\}$. If $x_f^t = 0$, then the content is not cached in the local cache, and $x_f^t = 1$ otherwise.

$$\sum_{f \in F} x_f^t \kappa_f \leqslant s, \tag{1}$$

TABLE 1. Table of notations.

Notation	Description
s	Cache size of the base station
$\mathcal{F}, F, f$	Set, number of, and elements of contents
κ_f	Size of the content f
r^t, r_f^t	Number of incoming requests for all content and content f respectively, at time t
$\{d^t \| t = 1, 2, ..., T\}$	Dataset of base station
$\boldsymbol{y}_j^t$	Input vector
$\pi_f^t, \hat{\pi}_f^t$	Real and predicted popularity classes, respectively, of content f at time t
$\phi_f^t, \hat{\phi}_f^t$	Real and predicted request counts, respectively, of movie f at time t
δ	Access delay or latency to retrieve per megabyte of movie's size from the content server
$x_f^t \in \{0, 1\}$	Cache decision variable at time t
$\kappa_{\text{delay}}(x_f^t)$	Content access delay for content f at time t
$\mathcal{M}_{\text{class}} = \{1, 2, 3, \ldots, M\}$	A set of class prediction models
$\mathcal{M}_{\text{req}} = \{1, 2, 3, \ldots, M\}$	A set of request count prediction models
l_m, γ_m	learning rate and regularization rate, respectively, of model m
$m_{\text{tacc}}, m_{\text{vacc}}, m_{\text{pacc}}$	training, validation, and prediction accuracy, respectively, of model m
ϵ	accuracy threshold
$a(z)$	activation function

where x_f^t is the cache decision control variable, κ_f is the size of the movie f, and s is the size of the BS's cache. Therefore, each BS has a limited cache capacity and cannot store more than its capacity. Note that, all of the cache miss contents will be retrieved from the content server.

B. CONTENT RETRIEVAL MODEL AND OBJECTIVE

Let $\kappa_{\text{delay}}^{t+1}$ denotes the delay of accessing content at the BS for the next time slot $(t + 1)$, given by:

$$\kappa_{\text{delay}}^{t+1}(x_f^t) = \sum_{f \in F} \left((1 - x_f^t) \delta \kappa_f \right) \phi_f^{t+1}, \tag{2}$$

where δ is the access delay or latency to retrieve the file size κ_f from the content server, and ϕ_f^{t+1} is the total number of request counts for content f at future time $t + 1$. Then, the future access delay at the BS is related to the current time t cache decision x_f^t, where the future request counts for the contents are unknown. In other words, we only have the prior request counts. Hence, to satisfy (3), we perform request count prediction, which will be discussed in Sec. V.

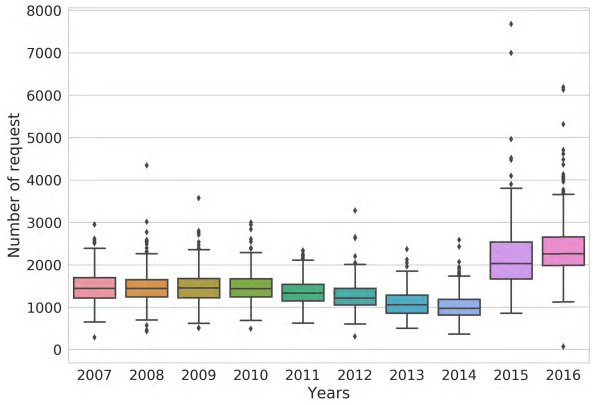

FIGURE 3. Daily request count of all contents (movies) for each year.

C. OPTIMIZATION PROBLEM

We formulate the optimization problem as minimizing the user access delay for contents, which is defined as follows:

$$\begin{aligned} &\underset{x_f^t}{\text{minimize:}} \ \frac{1}{T}\sum_{t\in T}\kappa_{\text{delay}}^{t+1}(x_f^t) \\ &\text{subject to: (1), (2),} \end{aligned} \tag{3}$$

where the objective function is minimized by controlling cache decision variable x_f^t at time t. In this problem, the future request counts for each content ϕ_f^{t+1} are not known in advance. Furthermore, to find the optimal solution, we need to exhaustively search the solution space, which is not practical due to the large number of configuration combinations. These two reasons render the optimization problem in (3) challenging to solve in the presence of limited information. Hence, to solve (3), we need a two-step solution: i) predicting $\hat{\phi}_f^{t+1}$, the expected request count of content f (Sec. IV-F), and ii) implement a caching decision (Sec. V).

IV. PREDICTING THE EXPECTED REQUEST COUNT

In this section, we discuss how to apply a deep learning model to predict the expected request count ϕ_f^{t+1} in order to support the cache decision. First, we perform exploratory data analysis on the Movie Lens dataset, which is used for training, validating and testing in order to measure the performance of the models. The dataset includes 26, 000, 000 ratings applied to 45, 000 movies by 270, 000 users. We assume that the rating counts are the same as the request counts of the movies, and that the times when users rate movies are same as the request arrival times for those movies. Also, this dataset includes other information related to the users such as age, sex, and occupation. Furthermore, the dataset contains movie related information such as release date and genre. We use *movie-ID*, *genre*, *year*, *day of year*, *month*, and *day* as categorical variables. Then, we use *request* information as numerical variable.

First, the daily requested movie counts are analyzed to determine how many movies are requested each day. Due to the limited space, only the results from 2007 to 2016 are shown in Fig. 3 with box plot, where the daily average number of requested movies for each day of the year is around 2, 000 movies. For the results, the minimum number of request arrivals for each day is around 500 movies and the maximum number of request arrival is around 7, 500. Therefore, all of the movies requested each day cannot be stored in the cache storage of the BS, where the cache storage size of the BS is relatively smaller than the number of daily requested movies.

Therefore, we extract the information for each requested movie and sort the contents in the descending order, as shown in Fig. 4, to determine the number of received requests based on the ranks of the movies. In Figs. 3-6, we found that the total number of received requests for each content based on the ranks of the movies follows a long-tailed distribution (such as a Zipf or power law distribution), where the movie requests are based on the ranks of the movies. Moreover, the shape

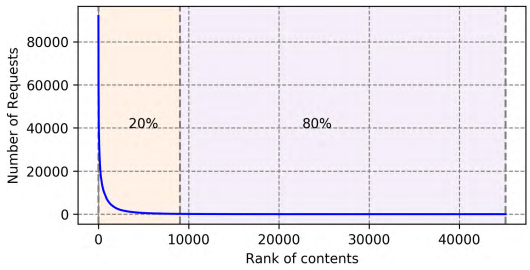

FIGURE 4. Ranking of all contents (movies) based on total request count received.

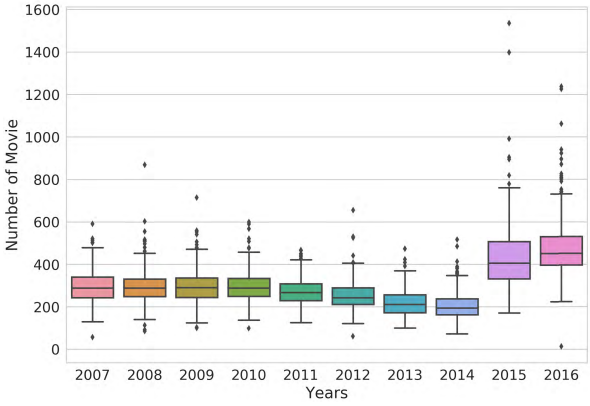

FIGURE 5. Daily request count of top 20% contents (movies) for each year.

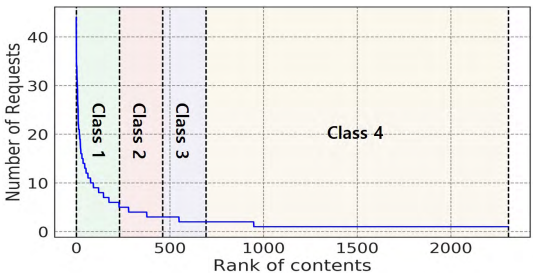

FIGURE 6. Ranking of contents (movies) based on daily received content requests and classification labels.

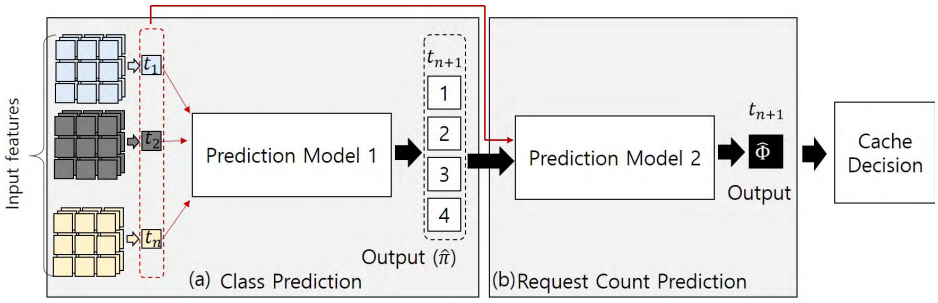

FIGURE 7. The overview prediction model used in the proposed scheme: (a) Class prediction, (b) Request count prediction.

of the long-tailed distribution changes at every time t (e.g., day). Note that the Zipf distribution is the discrete form of the Pareto distribution, which gives rise to the well-known Pareto principle.[3] Based on that finding, if we store the top 20% of popular movies, the traffic will be reduced by 80%.

As shown in Fig. 5, we extracted the daily requested movies count information for 20% of movies, where the daily average requested movie count is around 300 to 400 movies. Based on the results, the minimum number of request arrivals for each day is around 50 movies and the maximum is around 1, 500. Therefore, not all of the 20% of requested movies can be fit into the BS cache space.

Thus, we divided the requested movie counts into four categories, as shown in Fig. 6: Class 1 (Popularity level 1), Class 2 (Popularity level 2), Class 3 (Popularity level 3), and Class 4 (Popularity level 4), respectively. Based on these results, we wish to store Class 1, Class 2 and Class 3 movies, assuming the cache storage of the BS is large enough. We avoid storing Class 4 movies because those have fewer requests. Otherwise, we store movies based on the priority Class, where Class 1 is the highest priority. If the cache space is too small, even for Class 1 movies, we consider storing movies based on request count priority. Note that Class 5 contents received zero requests and is not shown in Fig. 6. Finally, our prediction model considers two prediction steps: i) predict the class labels (classification problem) and ii) predict the request count for each content based on the predicted class label (regression problem). In the next section, we discuss various types of possible models that can be used for two-step prediction.

Discussion: Note that we can directly predict the content popularity from the entire data set of the request counts as depicted in Figs. 3–4. However, it is not efficient because we have to process the entire content database. In addition to high computation cost, the prediction accuracy is low. Hence, we perform a classification to filter out the top 20% of the contents as shown in Fig. 5. Next, we perform prediction on the filtered top 20% content which also improves the prediction accuracy. The entire process is shown in Fig. 7.

[3]i.e., the 80-20 rule where the top 20% of the movies received 80% of the requests.

A. DEEP LEARNING MODELS

In this section, we first discuss the prediction model used in this paper, then, we present the various types of possible deep learning models to determine the best-suited models for our proposed scheme. The high-level prediction model design used in this paper is shown in Fig. 7, where the future request count is predicted based on the predicted class of the contents. Then, the predicted results are used to cache the contents using the Cache Decision Module. Thus, the prediction model is composed of two major parts: i) a class prediction model and ii) a request count prediction model. The input for class prediction is the tensor $\boldsymbol{y}$ (i.e., the multidimensional features) and the output is the predicted class label π. The input for request count prediction is the tensor $\boldsymbol{y}$, which includes the same features as for class prediction with extra feature (the predicted class label π). The output is the predicted request count ϕ. Next, we present the candidate models to choose for prediction model 1 and model 2 as follows.

1) RECURRENT NEURAL NETWORK

Since Recurrent Neural Networks (RNNs) are specifically designed for the temporal dynamics of input sequential data, they are chosen as a candidate model. Among various types of RNNs such as Gated Recurrent Units and Long Short-Term Memory (LSTM), we choose LSTM because neural networks using LSTM cells have offered better performance than standard recurrent units, such as in speech recognition [20]. LSTMs extend RNNs with memory cells to store and output information, helping with learning of temporal relationships over long timescales. Each LSTM cell includes a *forget gate*, *cell state*, *external input gate*, *output* and *output gate* [31].

The full LSTM-based prediction model used in this paper is shown in Fig. 8, which includes input layers, a batch normalization layer [32], recurrent layers, dropout, time-distributed dense network layers, and output layers. Batch normalization allows for much higher learning rates. Recurrent layers learn the time series data. Dropout is used to prevent the overfitting problem. The time-distributed dense network layer is used to classify the data.

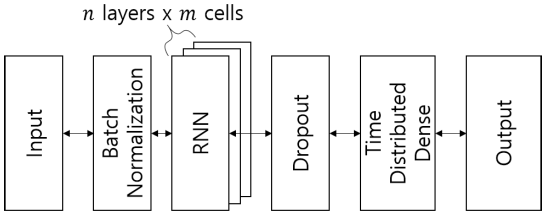

FIGURE 8. Recurrent neural network model.

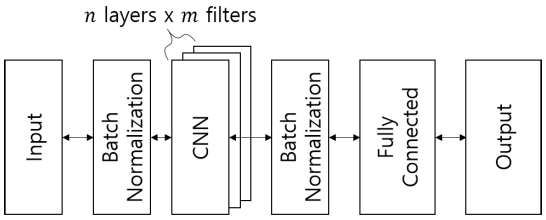

FIGURE 9. Convolutional neural network model.

2) CONVOLUTIONAL NEURAL NETWORK

Convolutional Neural Networks (CNNs) perform convolutions operation on the multi-dimensional input data to extract information and hence it is chosen as a candidate model. The main purpose of CNNs is to process data that has a known grid-like topology to discovering important features [33]. A typical CNN consists of a convolution layer, pooling layer and fully-connected layer. The *Convolutional layers* apply a convolution operation to the input and pass the results to the next layer. The *pooling layer* performs a downsampling operation along the spatial dimensions. Neurons in the *fully-connected layer* have full connections to all activations in the previous layer, as seen in the feed-forward neural network. Fig. 9 shows the full CNN-based prediction model used in this paper, which includes input layers, a batch normalization layer, convoutional layers, batch normalization, fully connected layers, and output layers. The convolutional layers allow us to convolute the information in neighboring contents to extract feature representations more efficiently. The other layers are the same as in the Recurrent Neural Network section.

3) CONVOLUTIONAL RECURRENT NEURAL NETWORK

A Convolutional Recurrent Neural Network (CRNN) model contains both convolutional layers and recurrent layers, which are capable of automatically learning feature representations and modeling the temporal dependencies between their activation. Among the various types of RNNs, we choose Long Short-Term Memory (LSTM) as a candidate model. The convolutional layers act as feature extractors and provide abstract representations of the multi-dimensional input data in the feature maps. The recurrent layers learn the temporal dynamics of output data from the convolutional layers. Fig. 10 shows the full Convolutional Recurrent Neural Network-based prediction model used in this paper, which includes the input layers, batch normalization layer,

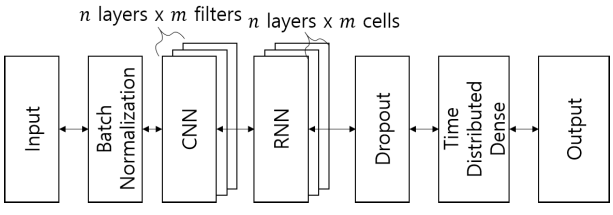

FIGURE 10. Convolutional recurrent neural network model.

recurrent layers, dropout, time-distributed dense network layers, and output layers.

B. DEFINING THE OUTPUT LAYERS OF DEEP LEARNING MODELS

In this section, we discuss various types of activation functions that can be used as output layers based on the prediction problem.

1) POPULARITY CLASS PREDICTION PROBLEM

The popularity class prediction problem is a kind of multi-class classification problem. Therefore, we used a Softmax layer as the final output layer, which is often used as the final layer for deep neural networks for multi-class classification to obtain the final output, which can be viewed as a probability of each popularity class label. The softmax function is defined as follows:

$$a(z_i) = \mathrm{sm}(z_i) = \frac{e^{z_i}}{\sum_j e_j^z}, \tag{4}$$

where sm is the softmax function that transforms a vector z with arbitrary real values to a vector with values ranging from 0 to 1.

2) REQUEST COUNT PREDICTION PROBLEM

The request count prediction problem is a kind of regression problem. Therefore, we have multiple choices for the activation function for the final output layers, such as sigmoid, tanh, and Relu. The sigmoid function can be defined as follows:

$$a(z_i) = \sigma(z_i) = \frac{1}{1 + e_i^{-z}}, \tag{5}$$

where the output value of the sigmoid function is between 0 and 1. The hyperbolic tangent function can be denoted as follows:

$$a(z_i) = \tanh(z_i) = \frac{e_i^z - e_{[i]}^{-z}}{e_i^z + e_i^{-z}}, \tag{6}$$

where the output value is between -1 and 1. The ratified linear unit can be denoted as follows:

$$a(z)_i = \mathrm{Relu}(z_i) = \begin{cases} 0, & \text{for } z_i < 0. \\ z, & \text{for } z_i \geqslant 0. \end{cases} \tag{7}$$

where, if the input is less than zero, the output of the relu function is 0.

C. LOSS FUNCTION

Based on the prediction problem, we also need to choose suitable loss function to minimize the prediction loss.

1) POPULARITY CLASS PREDICTION PROBLEM

The popularity class prediction is the multi-class classification problem. Therefore, we use categorical cross-entropy, a generalization of binary cross-entropy, as a loss function to determine the multi-class logarithmic loss. It is also referred to as negative log likelihood, which is used when a probabilistic interpretation of the prediction accuracy scores is desired. Let $\boldsymbol{\pi} = [\pi_1, \ldots, \pi_n]$ be a vector representing the true multinomial distribution over labels $1, \ldots, n$, and let $\hat{\boldsymbol{\pi}} = [\hat{\pi}_1, \ldots, \hat{\pi}_n]$ be the linear classifier's output, which was transformed by the softmax function. The categorical cross entropy loss measures the miss-classification between the true label $\boldsymbol{\pi}$ and the predicted label $\hat{\boldsymbol{\pi}}$, and is defined as the cross-entropy [33]:

$$L_(\hat{\boldsymbol{\pi}}, \boldsymbol{\pi}) = -\sum_i \pi_i \log(\hat{\pi}_i). \tag{8}$$

2) REQUEST COUNT PREDICTION PROBLEM

The request count prediction is a kind of regression problem. Therefore, we apply the Mean Squared Error (MSE) to measure loss because the mathematical properties of MSE makes it easier to compute the gradient. The MSE function can be denoted as follows:

$$L_(\hat{\boldsymbol{\phi}}, \boldsymbol{\phi}) = \frac{1}{n}\sum_{i=1}^{n}(\phi_i - \hat{\phi}_i)^2, \tag{9}$$

where ϕ_i is the real label and $\hat{\phi}_i$ is the predicted label.

D. COMPUTING GRADIENT

There are several optimizers for computing gradients, such as stochastic gradient descent, gradient descent and ADAM (adaptive moment estimation). Among them, based on the performance comparison of the various optimizers in [34], we choose ADAM [35], which provides better performance than the others. For computing the gradient of CNN, we used a backpropagation algorithm [36] with an ADAM optimizer. For computing the gradient of LSTM and CRNN, we used a Truncated Backpropagation Through Time (TBTT) method with an ADAM optimizer. TBTT is a modified version of the Backpropagation Through Time (BPTT) training algorithm for recurrent neural networks where the sequence is processed one-time step at a time and the BPTT update is performed backward for a fixed number of time steps.

E. MODELS GENERATING AND FINDING PROCESS

We find the best suited models for popularity class prediction and request count prediction among the three models (CNN, LSTM, and CRNN) with various settings for the hyperparameters, such as the number of layers and number of LSTM cells. As the name suggests, deep learning involves multiple layers, where each layer has multiple cells. Thus, each learning model can have a huge number of configuration combinations. Finding suitable models among the three models with different hyperparameters is a combinatorial problem. To search the models, we can use two different methods: i) grid-search and ii) random-search. Grid-search finds the best model by increasing the hyperparameter values in sequential combinations. As the name suggests, random-search finds the best model among random configurations in order to reduce search space. To reduce computational complexity, we choose random-search and randomly configure the hyperparameters to construct the models. Finding the optimal learning model for a given problem is outside the scope of this paper but we will explore this in the future.

Algorithm 1 Finding the Best Suited Prediction Model

Input: Small dataset $d^{[t=1:t=j]}$, num_model, model_type
Output: Best model $\boldsymbol{m}^*$ with μ_m^*, ν_m^*, η_m^*, θ_m^*.

1: Initialized $\mu_{\min}$, $\mu_{\max}$,$\nu_{\min}$, ν_{max}, $\eta_{\min}$, $\eta_{\max}$, $\theta_{\min}$, $\theta_{\max}$, $m_{\text{tacc}}, m_{\text{vacc}} = 0$, his $=\{\}$, l_m, $\gamma_m = 0.001$
2: **for** cur_model_type $\leftarrow$ model_type **do**
3: **for** k in range(num_models) **do**
4: **if** current model type is LSTM **then**
5: $m \leftarrow$ model_gen($\eta_{\min}$, $\eta_{\max}$, $\theta_{\min}$, $\theta_{\max}$, l_m, γ_m)
6: **else if** current model type is CNN **then**
7: $m \leftarrow$ model_gen($\mu_{\min}$, $\mu_{\max}$, $\nu_{\min}$, $\nu_{\max}$, l_m, γ_m)
8: **else**
9: $m \leftarrow$ model_gen($\mu_{\min}$, $\mu_{\max}$, $\nu_{\min}$, $\nu_{\max}$, $\eta_{\min}$, $\eta_{\max}$, $\theta_{\min}$, $\theta_{\max}$, l_m, γ_m)
10: Train the constructed model m with small dataset $d^{[t=1:t=j]}$
11: Store all of training information of each model m in array $his\{\} \leftarrow \{m : h_m, c_m, ls_m, m_{\text{tacc}}, m_{\text{vacc}}\}$
12: Choose the model $m^* = \arg\max(m_{\text{vacc}})$.

ALG. 1 is the model generation and selection algorithm, where the inputs of the algorithm are the features and target labels from the small dataset $d^{[t=1:t=j]}$, number of model to be constructed, and the type of the models (e.g., LSTM, CNN, etc.). The output is the best model m^* with the optimal hyperparameters μ_m^*, ν_m^*, η_m^*, θ_m^*.

First, the searching space for the hyperparameters are initialized in line 1 as follows:

- The minimum and maximum numbers of convolutional filters ($\mu_{\min}$ and $\mu_{\max}$).
- The minimum and maximum numbers of convolutional layers ($\nu_{\min}$ and $\nu_{\max}$).
- The minimum and maximum numbers of LSTM cells ($\eta_{\min}$ and $\eta_{\max}$).
- The minimum and maximum numbers of LSTM layers ($\theta_{\min}$ and $\theta_{\max}$).
- The learning rate l_m and regularization rate γ_m with value 0.001.
- The number of models to be generated num_models.

Algorithm 2 Training Process

Input: model m^*, train dataset $d^{[t=1:t=j]}$
Output: Trained model m^*

1: Initialized $m_{\text{tacc}}, m_{\text{vacc}}, m_{\text{pacc}} = 0, l_m, \gamma_m = 0.001$
▷ **Find best parameters**
2: **for** i in range(n) **do**
3: $l_{m,i}, \gamma_{m,i} \leftarrow$ rand(0, 1)
4: Train the model m with small dataset by minimizing loss
5: **if** $m_{vacc}^{i-1} < m_{vacc}^{i}$ **then**
6: $l_m^*, \gamma_m^* \leftarrow l_{mi}, r_{mi}$
7: Reset model m
▷ **Train model**
8: Train model m with l_m^*, γ_m^*.
9: **while** $m_{\text{tacc}} < \varepsilon$ or dataset is not fully utilized **do**
10: Minimize the training loss of model m based on the loss function from Sec. (IV-C) with the dedicated optimizer from Sec. (IV-D)
11: **if** Current validation accuracy of model m_{tacc} is better than the previous **then**
12: Update and store the parameters of m

Then, the various type of prediction models are constructed based on the process from lines 2 to 9. We check the number of models to be generated in line 3. Then, we construct the desired model LSTM, CNN or CRNN as described in line 5, 7 or 9, respectively. Then we train each constructed model with a small dataset (line 10). The historical training information of each model m are stored in *his* array (line 11), which includes the number of hidden layers h_m, convolutional filters c_m, LSTM cells η_m, training accuracy m_{tacc}, validation accuracy m_{vacc}. Finally, we choose the model which has the maximum validation accuracy value (line 12).

F. TRAINING ALGORITHM

Alg. 2 handles the training procedures for the prediction model at the cloud data center to improve the accuracy of the prediction model. The inputs for this algorithm are the best model m^* and the subset of raw data $d^{[t=1:t=j]}$. The output of this algorithm is an optimized trained model for predicting popularity scores. First, the accuracy measurement metrics $m_{\text{tacc}}, m_{\text{vacc}}, and m_{\text{pacc}}$ are initialized with the value zero. Then, the learning rate l_m and regularization rate γ_m are initialized with fixed values of 0.001 (line 1).

Second, we find the best values for the learning rate l_m and regularization rate γ_m (from lines 2 to 7). We define the number of rounds n to find the parameters l_m and γ_m (line 2). For every round i, the values of $l_{m,i}$ and $\gamma_{m,i}$ are initialized with a uniform random generator (line 3). Training model m with a small dataset is performed to compare the validation accuracy (line 4). If the old validation accuracy m_{vacc}^{i-1} is less than the current accuracy m_{vacc}^{i} (line 5), then store the current learning rate and regularization rate (line 6). At the end of every round i, reset the model (line 7).

Algorithm 3 Cache Decision Process

Input: Contents request history
Output: Contents list to store

▷ **Master Node**
1: Predict the future $(t+1)$ popularity class and request counts of video contents by using trained model m for BS
2: **if** $m_{\text{pacc}} < \varepsilon$ **then**
3: Repeat lines 9 to 13 from Alg. 2 with daily data d_{bs}^{t+1} to update parameters of model m such as weight **w**
4: Send the predicted information to *Slave Node*
▷ **Slave Node**
5: Received the predicted information
6: **if** Total size of Class 1 contents < s **then**
7: Store all of Class 1 contents
8: **if** Cache space s still has free space **then**
9: Sort Class 2 contents in descending ordered based on predicted request count
10: Store the contents from the beginning until the cache space is full
11: **if** Cache space s still has free space **then**
12: Sort Class 3 contents in descending ordered based on predicted request count
13: Store the contents from the beginning until the cache space is full
14: **else**
15: Sort Class 1 contents in descending ordered based on predicted request count
16: Store the contents from the beginning until the cache space is full

Third, the process of training model m with the complete dataset is presented in lines 8 to 12. Then, the optimal learning rate l_m^* and regularization rate γ_m^* are initialized to start the training processes. Next, the model m is trained until the validation accuracy reaches a certain accuracy level ε (lines 9 to 12). If the validation accuracy of the current training is better than the old one, all of the updated hyperparameters settings are stored (lines 11 and 12).

V. CACHE DECISION

Alg. 3 shows the caching process at the BS to improve the cache hit. The inputs for this algorithm are the requests from users, the log files from the BS and the trained models from the cloud data center, which are used to predict the popularity scores. The output of this algorithm is the decision on whether to store the expected popular contents. The caching decision process includes two main parts: the *Master Node* side (from lines 1 to 4) and the *Slave Node* side (from lines 5 to 16). The *Master Node* predicts the future class and future popularity of contents by using the trained model m generated from ALG. 2 (line 1). If the prediction accuracy is lower than the threshold, the model is trained with daily data (lines 2 and 3) and the predicted information is sent to the *Slave Node* (line 4).

After receiving the predicted results, the *Slave Node* checks the total size of contents in Class 1. First, BS sort the Class 1 contents based on predicted request count. Next, BS stores Class 1 contents until its cache space is full (line 14 to 16). If there are remaining cache space, the process is repeated for Class 2 and Class 3 contents (line 8 to 13).

VI. PERFORMANCE EVALUATION

In this section, we first discuss how to measure the performance of our proposed scheme compared with others. Then, we discuss the detailed process of finding the best models for training and implementation. We run our tests on a PC with the followings configurations; CPU: Intel core i7-7700k, RAM: 32GB, graphics card: GeForce GTX 1080 Ti, and operating system: Ubuntu 16.04 LTS. To preprocess the dataset, we used pandas [37], which is an open source data analysis tool. We developed prediction models using the GPU version of Tensorflow 1.4 [30] as a backend and Keras [29] as a frontend which are both open source software libraries for machine learning.

A. PERFORMANCE MATRICES

We choose the *categorical cross-entropy* and *Mean Squared Error* (MSE) to measure the accuracy of the prediction models. To measure the performance of cache decision, we choose *backhaul usage*, *probability of hit*, and *access delay* as the Key Performance Indicator (KPI) metrics.

1) PREDICTION ACCURACY

For the classification problem, we use Categorical cross-entropy to find the multi-class logarithmic loss as in (8). For the request prediction, we apply the MSE to measure the loss, as in (9).

2) CACHE DECISION RELATED METRICS

a: PROBABILITY OF HIT

Typically, the performance of the cache is measured in terms of the probability that the requested chunks are found at a given content store. The probability of a cache hit measures the hit and miss probabilities to determine how much traffic the network can eliminate. The probability of a cache hit is calculated as follows:

$$P_{\text{hit}} = \frac{\sum_{t\in T}\sum_{f\in\mathcal{F}} x_f^t r_f^t}{\sum_{t\in T}\sum_{f\in\mathcal{F}} r_f^t}, \tag{10}$$

where P_{hit} is the probability of hits, $\sum_{t\in T}\sum_{f\in F} r_f^t c_f^t$ is the total number of hits at the routers, BSs and SBSs, and $\sum_{t\in T}\sum_{f\in F} r_f^t$ is the total number of cache hits and cache misses.

b: ACCESS DELAY

In addition, we measure the latency of downloading the contents from the user side, which is calculated as in (2).

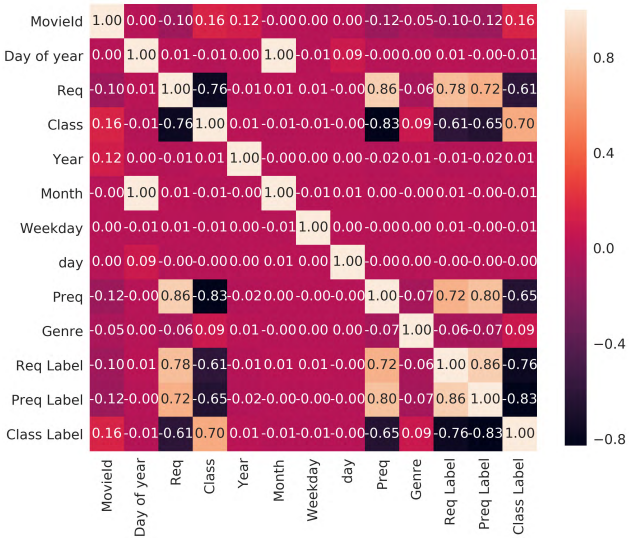

FIGURE 11. Features correlation matrix.

c: BACKHAUL USAGE

In addition, we measure the backhaul usage, which is calculated as follows:

$$P_{\text{bachhaul}} = \sum_{f\in F}\kappa_f \sum_{t\in T} r_f^t \tag{11}$$

where κ_f is the size of the content f, and r_f^t is the number of arrival requests for content f at time t.

B. FINDING THE BEST PREDICTION MODEL

In this section, the process of finding a suitable prediction model is discussed in detail. First, finding the important features to train is presented. Second, finding the optimal number of neighbor information data to train is discussed. Third, finding the optimal time slot information is explained. Finally, the models we used in training are discussed.

1) FEATURE SELECTION

In this section, we find the best features to train the prediction models. In other word, we do the feature selection, which is a process to select features that are most related to the prediction outputs. Three benefits of performing feature selection are as follows: i) reduce Overfitting, i.e., less redundant data means less opportunity to make decisions based on noise; ii) improve Accuracy, i.e., less misleading data means the modeling accuracy improves; and iii) Reduces Training Time, i.e., fewer data means that the algorithm train faster.

Table 2 shows the details of the extracted features from the dataset and Table 3 shows the scoring methods to choose the important features. Fig. 11 shows the results of the Pearson product moment correlation coefficient matrix as a heat map. Then, in Fig. 12, we compare the importance of features in terms of different scoring methods such as information gain. From Fig. 11 and Fig. 12, we can clearly see which features are the most relevant for our training algorithm

TABLE 2. Table of features.

Features	Description
MovieID	Movie identification number from 1 to 38012.
Day of year	Counting the number of days from 1 to 365. Note that leap year has 366 days.
Req	Request arrival counts of each movie for each day.
Class	Class number of movie from Class 1 to 5.
Year	Year information (e.g., 2010,2011, etc.).
Month	Month information which is encoded as the number from 1 to 12.
Weekday	Week information which is encoded as the number from 1 to 7.
Preq	Request arrival counts in probability.
Genre	Genre information of each movie such as "Action" movie, which is encoded as the number from 1 to 19.
Req Label	Received request counts in the next day.
Preq Label	Probability of received request counts in the next day.
Class Label	Class label of movie for the next day.

TABLE 3. Table of scoring methods.

Scoring Methods	Description
Pearson	Pearson product moment correlation coefficient measures the linear dependence between pairs of the input features and target labels [39].
Info. Gain	Information Gain measures the expected amount of information (reduction of entropy). [40]
Gain ratio	An Information gain ratio measures a ratio of information gain related to the intrinsic information of features to defeat a bias towards multi-valued characteristics. [40]
Gini	Gini coefficient is used to measures the inequality between values of a frequency distribution. [41]
ANOVA	ANOVA stands for Analysis of variance and it measures the difference between average values of the feature in different classes. [42]
χ^2	Chi-square measures dependence between the feature and the class as measured by the chi-square statistic. [42]

Features	Info. gain	Gain ratio	Gini	ANOVA	χ^2	Univar. reg.
Req	0.12142997	0.2511398	0.026886	4407015.5	19620938	24240353.06
Preq	0.11616168	0.2324513	0.025013	5691412.5	18874553	39504003.25
Class	0.09919853	0.2597476	0.020331	5610326.1	461164	16513038.44
MovieId	0.02465540	0.0123277	0.003242	188394.9	580894	301683.94
Genre	0.00762264	0.0039809	0.000815	42640.0	171818	115008.91
Weekday	0.00032090	0.0001645	0.000046	1008.6	5565	1.58
Month	0.00012563	0.0000628	0.000018	811.0	3045	11.22
Day of year	0.00012469	0.0000623	0.000018	801.4	2989	15.83
Year	0.00011787	0.0001187	0.000017	914.5	1648	6182.53
Day	0.00000521	0.0000026	0.000001	1.5	22	15.75
Class Label						48528301.91

FIGURE 12. Features ranking for the class label prediction (white) and request counts prediction (orange), where the color bars represent the level of importance for each feature.

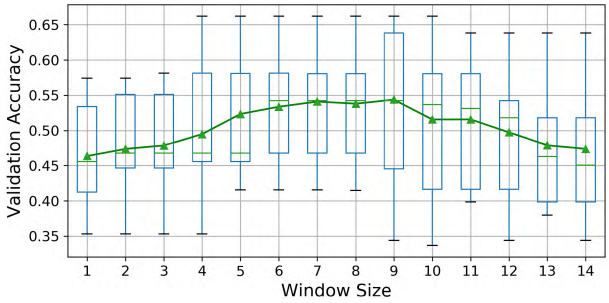

FIGURE 13. Finding a suitable number of sequences data.

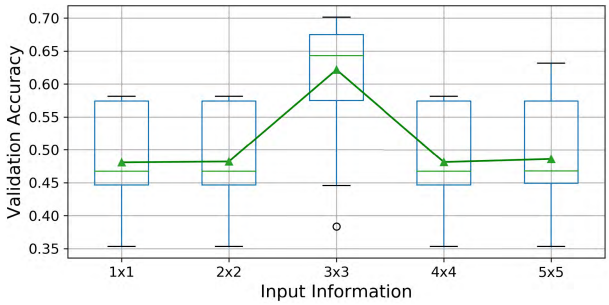

FIGURE 14. Prediction accuracy based on the neighbors' information.

based on different measurement metrics. We choose Req, Preq, Class, MovieId, and Genre for class label prediction. Similarly, we choose Req, Preq, Class, MovieId and Class Label (also known as the predicted Class Label) for request count prediction.

2) FINDING SUITABLE TIME SEQUENCE INFORMATION

Next, we determine the best number of continuous sequence data or window size to improve the accuracy and reduce the training time of the prediction model. In this best suited window size finding process, we simply use the LSTM model which has 3 LSTM layers, and each layer has 3 LSTM cells. We also fixed the learning rate to 0.01 and the regularization rate to 0.01. Then we feed the 1 month worth data with the window size varies from 1 to 14, and the results are shown in Fig. 13. Generally, when the number of sequences increased, the time needed to process data is increased. The results from Fig. 13 show that the validation accuracy is increasing from window size 1 to 7, and then the accuracy is decreasing from window size 8. Thus, based on these results, we choose window size 7 to train the prediction model.

3) FINDING THE INPUT GRID SIZE

Fig. 14 clearly shows how the input grid affects the validation accuracy level. Hence, we need to carefully determine the input grid size (including the movie we want to perform prediction and its neighbors) to achieve high accuracy. For this experiment, we test with the same LSTM model and same configuration used in Sec. VI-B.2. As shown in Fig. 14, the validation accuracy is the highest for input grid size 3×3 which is used for predicting popularity class and request counts.

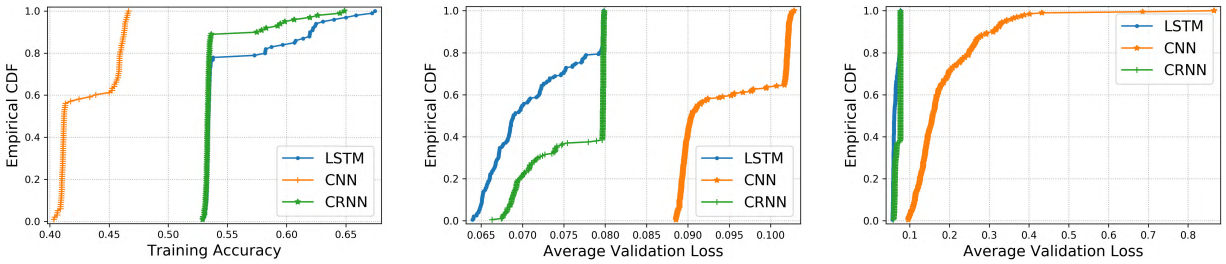

FIGURE 15. Class prediction performance comparison between LSTM, CNN, and CRNN: (a) average training accuracy, (b) average validation accuracy, and (c) average computation time.

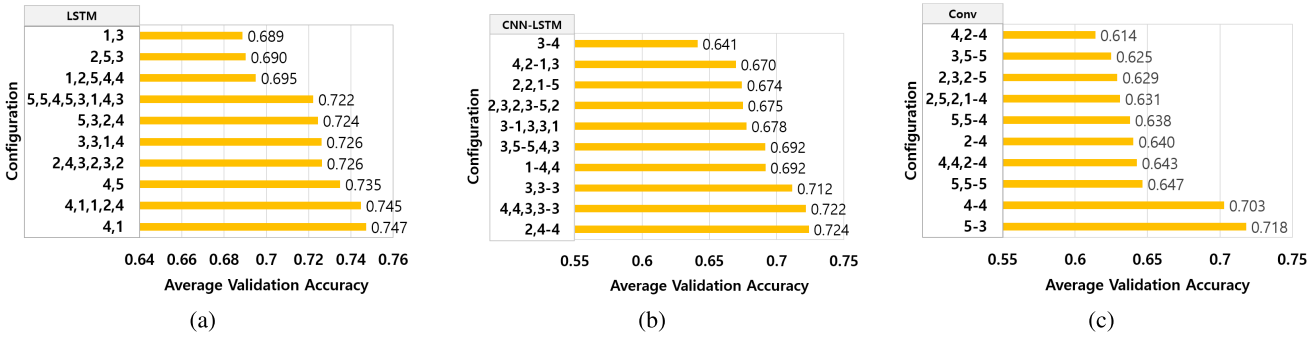

FIGURE 16. Class prediction validation accuracy for the best 10 out of 100 randomly chosen deep learning model configurations (a) LSTM, (b) CRNN, and (c) CNN.

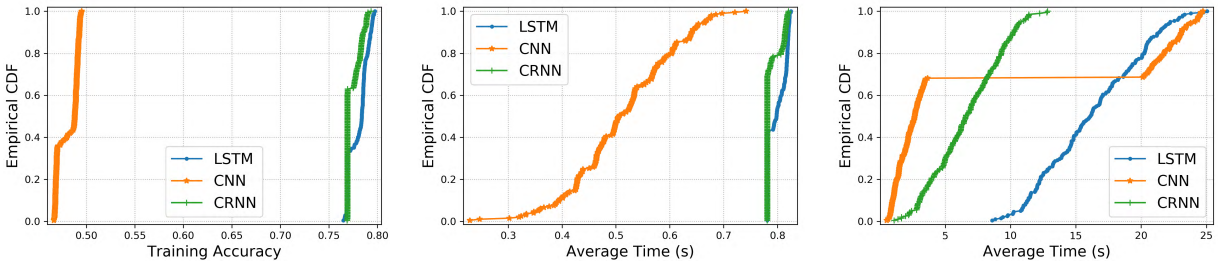

FIGURE 17. Request counts prediction performance comparison between LSTM, CRNN, and CNN: (a) average training accuracy, (b) average validation accuracy, and (c) average computation time.

4) THE MODELS USED FOR PREDICTION

To find the best model, we utilized Algorithm 1. We initialized the hyperparameters and configurations as shown in Table 4. Note that, in this paper, because of the page limitation, we only show the results which use the Relu activation function as the output layer of the request counts prediction model because Relu gives the best accuracy compared to others. Next, we randomly generate 100 configurations for each deep learning architecture. Then, we use one month of data as input with window size 7 and input grid size 3×3 to train the generated deep learning models. We plot the results of those 100 randomly generated deep learning model configurations for each of the three architecture in Figs. 15–18.

Fig. 15 shows the class prediction performance in terms of (a) average training accuracy, (b) average validation accuracy, and (c) average computation time for the three deep learning architectures, i.e., LSTM, CNN, and CRNN. From Fig. 15, we can clearly see that LSTM models outperforms those of both CNN and CRNN architectures on all key metrics, i.e., in terms of training accuracy, validation accuracy, and training (computation) time. We only plot the class prediction validation accuracy of the best 10 out of 100 randomly chosen deep learning models in Fig. 16 because validation accuracy is the most important parameter among others. From Fig. 16, we can clearly see that configuration LSTM 4,1, (i.e., LSTM model with 4 cells in layer-1 and 1 cell in layer-2), has the best validation accuracy and thus is chosen for our class prediction algorithm.

Fig. 17 shows the request counts prediction performance in terms of (a) average training accuracy, (b) average validation accuracy, and (c) average computation time for LSTM, CNN,

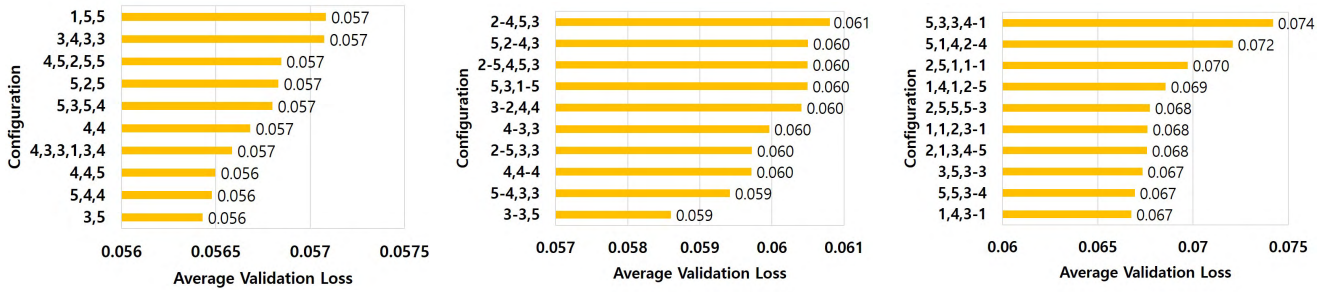

FIGURE 18. Request count prediction validation accuracy for the best 10 out of 100 randomly chosen deep learning model configurations (a) LSTM, (b) CRNN, and (c) CNN.

TABLE 4. Parameters used in the experiments.

Parameters	Symbol	Value
Min and max convolutional filters respectively	$(\mu_{\min}, \mu_{\max})$	$(1, 20)$
Min and max convolutional layers respectively	$(\nu_{\min}, \nu_{\max})$	$(1, 5)$
Min and max LSTM cells respectively	$(\eta_{\min}, \eta_{\max})$	$(1, 20)$
Min and max LSTM layers respectively	$(\theta_{\min}, \theta_{\max})$	$(1, 20)$
Learning and regularization rate	l_m, γ_m	0.001
num_models	m	600
Window size		7
Neighbors information (dimension)		3×3
Features (Class)		5
Features (Request)		6
Classification output size		4
Classification output activation function		Categorical cross-entropy
Request count output size		1
Request count output activation function		Relu
Batch Size		500
Training data		80% (2007-2015)
Validation data		20% (2007-2015)
Testing data		100% (2016)

TABLE 5. Comparison of prediction models.

Model Type	Configuration	Validation accuracy	Computing Time (s)
CRNN (Class)	Conv[2,4], LSTM[4]	0.643	7.258
LSTM (Class)	LSTM[4,1]	0.747	1.302
CNN (Class)	Conv[4,4,2], FC[1]	0.724	2.354
Model Type	**Configuration**	**Validation loss**	**Computing Time (s)**
CRNN (Req)	Conv[3], LSTM[3,5]	0.059	8.853
LSTM (Req)	LSTM[3,5]	0.056	17.547
CNN (Req)	Conv[1,4,3], FC[3]	0.066	22.160

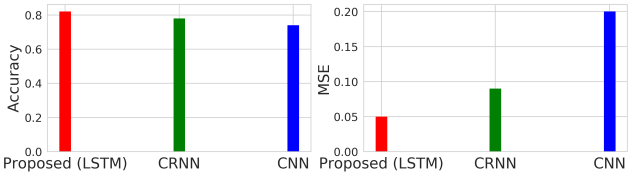

FIGURE 19. Average class label prediction accuracy (left). Mean squared errors of the request count prediction (Right).

and CRNN architectures. Fig. 17 shows that LSTM models outperforms both CNN and CRNN architectures on all key metrics, i.e., in terms of training accuracy, validation accuracy, and training (computation) time. We plot the validation accuracy of the request counts prediction for the best 10 out of 100 randomly chosen deep learning models in terms of in Fig. 18. Fig. 18 shows that configuration LSTM 3,5, (i.e., LSTM model with 3 cells in layer-1 and 5 cell in layer-2), has the best validation accuracy and thus is chosen for our request count prediction algorithm. Then, we summarize the best model in each deep learning architecture in Table 5 for performance comparison.

C. RESULTS AND DISCUSSION

To measure the performance of the proposed scheme, we train the prediction models using data collected from 2007 to 2014. Then, we test the accuracy of the prediction models with the 2016 data, which has 366 days' worth of information where each day has on average 2500 unique content requests or data points with target labels. For this comparison, we choose the models shown in Table 5 and the optimal solution.

1) BENCHMARK

Since we have the complete data set of requests for all contents, we can create the *optimal scheme* where we cache the contents which will possibly receive the most requests. We also consider the *no cache scheme* where we do not deploy the cache space in the network. We will use this *no cache scheme* as a baseline scheme for the performance evaluation of our proposal. The overall performance evaluation in term of different Key Performance Indicator (KPI) is shown in Tab. 6. Intuitively, we know that bigger cache size will result in better performance for all models, which is validated in Fig. 20.

2) VALIDATION ACCURACY

We compute categorical cross-entropy to measure the prediction accuracy and plot the average class label prediction accuracy in Fig. 19 (left) for the three deep learning models. Among them, the LSTM model provides the most accurate

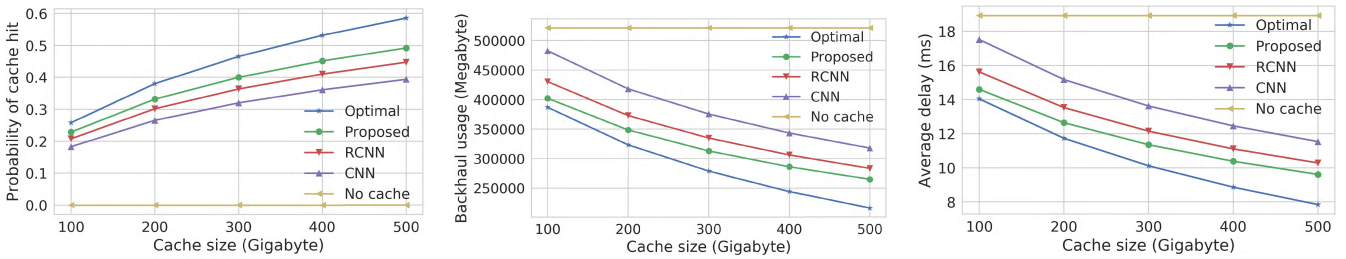

FIGURE 20. Caching related performance comparisons: (a) Cache hit, (b) Backhaul usage comparison, and (c) Access delay comparison.

TABLE 6. Performance evaluation of different KPI's compared to baseline (without cache scheme).

KPI	Optimal	LSTM	CRNN	CNN
Cache hit	+44%	+38%	+34%	+30%
Backhaul usage	-44.4%	-38%	-33%	-25%
Average delay	-42%	-36%	-31%	-24%

results for predicting the future class label. The request count prediction accuracy is measured using the MSE and the results are shown in Fig. 19 (right). Among the different types of models, the LSTM model provides the most accurate results. Due to Zipf distribution, Class 1 and Class 2 contents have the highest request counts. Thus, correctly predicting Class 1 and Class 2 contents will have the highest impact on caching as previously shown in Fig. 6.

3) PROBABILITY OF CACHE HIT

Next, we compare the cache hit probability for three different deep learning models in Fig. 20(a). A higher probability of hit means better performance. The results in Fig. 20(a) show that on average, the cache hit of the optimal scheme is improved by 44%, that of our proposed scheme is improved by 38%, that of the CRNN-based caching scheme is improved by 34%, and that of the CNN-based caching scheme is improved by 30% compared to benchmark (no cache). This means that our proposed model can predict the Class 1 and Class 2 contents better than other methods. Hence, our proposal can cache more Class 1 and Class 2 contents pro-actively based on better prediction results compared to others.

4) BACKHAUL USAGE

Fig. 20(b) shows the backhaul usage comparison, which is affected by the cache decision. Whenever a cache miss occurs, the request is passed to the content server which increases the traffic on the backhaul network. On the other hand, when the content is found in the cache, it is immediately replied to the user without any additional traffic on the backhaul network. Moreover, if a cache miss occur, a Class 1 content will have higher traffic than a Class 2 content on the backhaul network. Thus, it is imperative that Class 1 and Class 2 contents be classified with high accuracy. The results in Fig. 20(b) show that the backhaul usage of the optimal scheme is reduced by 44.4%, that of our proposed scheme is reduced by 38%, that of the CRNN-based caching scheme is reduced by 33%, and that of the CNN-based caching scheme is reduced by 25% compared to benchmark (no cache). Therefore, Fig. 20(b) validates that our algorithm can predict Class 1 and Class 2 better than others.

5) AVERAGE DELAY

Fig. 20(c) shows the average delay for retrieving content from user devices. As a result, the average delay to access contents is decreased when the prediction accuracy is increased, and a lower delay means a better performance. As the results, optimal scheme reduces the average delay by 42%, our proposed scheme reduces it by 36% , the CRNN-based caching scheme reduces it by 31%, and the CNN-based caching scheme reduces it by 24% compared to benchmark (no cache).

VII. CONCLUSION AND FUTURE WORK

In this paper, we proposed a learning-based caching method to improve the cache hit probability, backhaul usage, and video contents access delay. We proposed a two-step prediction algorithm which first classifies the video contents into four classes and then predicts the request counts. Our two-step prediction algorithm filters out the top 20% video contents which reduce computation cost and improve accuracy. This two-step algorithm uses the deep-learning model which has an infinite number of configurations. Typically, grid-search is used to find the best models, but this is a combinational problem and not easy to solve. Therefore, we instead find the best models based on the random-search method, in order to reduce the search space where we sample 100 configurations for each learning model. Then, we utilized the best models to predict the future class labels and request count prediction. Next, we proposed a caching scheme that utilizes the predicted class label and request count information to make cache decisions at the *Master Node*. This is accomplished by producing a list of contents that needed to be cached at the base station. The main challenges in this paper are determining suitable models among various types of deep learning models and configuring hyperparameters settings. Therefore, for future work, we will propose a scheme that autonomously finds out the best model more efficiently.

REFERENCES

[1] Accessed: Oct. 7, 2018. [Online]. Available: http://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html
[2] M. Zhang, H. Luo, and H. Zhang, "A survey of caching mechanisms in information-centric networking," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 3, pp. 1473–1499, 3rd Quart., 2015, doi: 10.1109/COMST.2015.2420097.
[3] D. Liu, B. Chen, C. Yang, and A. F. Molisch, "Caching at the wireless edge: Design aspects, challenges, and future directions," *IEEE Commun. Mag.*, vol. 54, no. 9, pp. 22–28, Sep. 2016, doi: 10.1109/MCOM.2016.7565183.
[4] T. Zhang, H. Fan, J. Loo, and D. Liu, "User preference aware caching deployment for device-to-device caching networks," *IEEE Syst. J.*, to be published, doi: 10.1109/JSYST.2017.2773580.
[5] A. Dabirmoghaddam, M. M. Barijough, and J. J. Garcia-Luna-Aceves, "Understanding optimal caching and opportunistic caching at 'the edge' of information-centric networks," in *Proc. 1st ACM Conf. Inf.-Centric Netw.*, Paris, France, Sep. 2014, pp. 47–56.
[6] K. Li, C. Yang, Z. Chen, and M. Tao, "Optimization and analysis of probabilistic caching in N-tier heterogeneous networks," *IEEE Trans. Wireless Commun.*, vol. 17, no. 2, pp. 1283–1297, Feb. 2018.
[7] E. Baştuğ, M. Bennis, and M. Debbah. (2014). "Living on the edge: The role of proactive caching in 5G wireless networks." [Online]. Available: https://arxiv.org/abs/1405.5974
[8] S. Ullah, K. Thar, and C. S. Hong, "Management of scalable video streaming in information centric networking," *Multimedia Tools Appl.*, vol. 76, no. 20, pp. 21519–21546, Oct. 2017.
[9] W.-X. Liu, J. Zhang, Z.-W. Liang, L.-X. Peng, and J. Cai, "Content popularity prediction and caching for ICN: A deep learning approach with SDN," *IEEE Access*, vol. 6, pp. 5075–5089, 2018, doi: 10.1109/ACCESS.2017.2781716.
[10] M. Chen, W. Saad, C. Yin, and M. Debbah, "Echo state networks for proactive caching in cloud-based radio access networks with mobile users," *IEEE Trans. Wireless Commun.*, vol. 16, no. 6, pp. 3520–3535, Jun. 2017, doi: 10.1109/TWC.2017.2683482.
[11] Z. Chang, L. Lei, Z. Zhou, S. Mao, and T. Ristaniemi, "Learn to cache: Machine learning for network edge caching in the big data era," *IEEE Wireless Commun.*, vol. 25, no. 3, pp. 28–35, Jun. 2018, doi: 10.1109/MWC.2018.1700317.
[12] R. Devooght and H. Bersini. (2016). "Collaborative filtering with recurrent neural networks." [Online]. Available: https://arxiv.org/abs/1608.07400
[13] K. C. Tsai, L. Wang, and Z. Han, "Mobile social media networks caching with convolutional neural network," in *Proc. IEEE Wireless Commun. Netw. Conf. Workshops (WCNCW)*, Apr. 2018, pp. 83–88.
[14] S. Ouyang, C. Li, and X. Li, "A peek into the future: Predicting the popularity of online videos," *IEEE Access*, vol. 4, pp. 3026–3033, Jun. 2016, doi: 10.1109/ACCESS.2016.2580911.
[15] S. He, H. Tian, and X. Lyu, "Edge popularity prediction based on social-driven propagation dynamics," *IEEE Commun. Lett.*, vol. 21, no. 5, pp. 1027–1030, May 2017, doi: 10.1109/LCOMM.2017.2655038.
[16] C. Li, J. Liu, and S. Ouyang, "Characterizing and predicting the popularity of online videos," *IEEE Access*, vol. 4, pp. 1630–1641, 2016, doi: 10.1109/ACCESS.2016.2552218.
[17] G. Szabo and B. A. Huberman, "Predicting the popularity of online content," *Commun. ACM*, vol. 53, no. 8, pp. 80–88, Aug. 2010.
[18] S. Li, J. Xu, M. V. D. Schaar, and W. Li, "Popularity-driven content caching," in *Proc. IEEE INFOCOM Conf.*, San Francisco, CA, USA, Apr. 2016, pp. 1–9, doi: 10.1109/INFOCOM.2016.7524381.
[19] R. Kleinberg, A. Slivkins, and E. Upfal, "Multi-armed bandits in metric spaces," in *Proc. 40th Annu. ACM Symp. Theory Comput.*, 2008, pp. 681–690.
[20] Q. Zhang, L. T. Yang, Z. Chen, and P. Li, "A survey on deep learning for big data," *Inf. Fusion*, vol. 42, pp. 146–157, Jul. 2018, doi: 10.1016/j.inffus.2017.10.006.
[21] M. Beyer and D. Laney, *The Importance of 'Big Data': A Definition*. Stamford, CT, USA: Gartner, 2012.
[22] E. Zeydan *et al.*, "Big data caching for networking: Moving from cloud to edge," *IEEE Commun. Mag.*, vol. 54, no. 9, pp. 36–42, Sep. 2016, doi: 10.1109/MCOM.2016.7565185.
[23] E. Baştuğ *et al.*, "Big data meets telcos: A proactive caching perspective," *J. Commun. Netw.*, vol. 17, no. 6, pp. 549–557, Dec. 2015, doi: 10.1109/JCN.2015.000102.
[24] J. Cobb and H. ElAarag, "Web proxy cache replacement scheme based on back-propagation neural network," *J. Syst. Softw.*, vol. 81, no. 9, pp. 1539–1558, 2008.
[25] S. Romano and H. ElAarag, "A neural network proxy cache replacement strategy and its implementation in the squid proxy server," *Neural Comput. Appl.*, vol. 20, no. 1, pp. 59–78, 2011.
[26] W. Tain, B. Choi, and V. V. Phoha, "An adaptive Web cache access predictor using neural network," in *Developments in Applied Artificial Intelligence, IEA/AIE* (Lecture Notes in Computer Science), vol. 2358, T. Hendtlass and M. Ali, Eds. Berlin, Germany: Springer, 2002, pp. 450–459.
[27] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[28] Accessed: Sep. 5, 2018. [Online]. Available: https://grouplens.org/datasets/movielens/
[29] Accessed: Sep. 5, 2018. [Online]. Available: https://keras.io/
[30] Accessed: Sep. 5, 2018. [Online]. Available: https://www.tensorflow.org/
[31] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[32] S. Ioffe and C. Szegedy. (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift." [Online]. Available: https://arxiv.org/abs/1502.03167
[33] T. Goldberg, *Neural Network Methods for Natural Language Processing*. San Rafael, CA, USA: Morgan & Claypool, 2017.
[34] S. Ruder. (2016). "An overview of gradient descent optimization algorithms." [Online]. Available: https://arxiv.org/abs/1609.04747
[35] D. P. Kingma and J. Ba. (2014). "Adam: A method for stochastic optimization." [Online]. Available: https://arxiv.org/abs/1412.6980
[36] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct. 1986.
[37] Accessed: Sep. 5, 2018. [Online]. Available: https://pandas.pydata.org/
[38] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[39] J. L. Rodgers and W. A. Nicewander, "Thirteen ways to look at the correlation coefficient," *Amer. Statist.*, vol. 42, no. 1, pp. 59–66, 1988.
[40] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986.
[41] C. Gini, "Concentration and dependency ratios," *Rivista Politica Econ.*, vol. 87, pp. 769–789, 1997.
[42] D. A. Freedman, *Statistical Models: Theory and Practice*. Cambridge, U.K.: Cambridge Univ. Press, 2005.

KYI THAR received the Bachelor of Computer Technology degree from the University of Computer Studies, Yangon, Myanmar, in 2007. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, Kyung Hee University, South Korea, for which he was awarded a scholarship for his graduate study in 2012. His research interests include name-based routing, in-network caching, multimedia communication, scalable video streaming, wireless network virtualization, deep learning, and future Internet.

NGUYEN H. TRAN (S'10–M'11) received the B.S. degree in electrical and computer engineering from the Hochiminh City University of Technology and the Ph.D. degree in electrical and computer engineering from Kyung Hee University in 2005 and 2011, respectively. He was an Assistant Professor with the Department of Computer Science and Engineering, Kyung Hee University, from 2012 to 2017. Since 2018, he has been with the School of Information Technologies, The University of Sydney, where he is currently a Senior Lecturer. His research interest is applying analytic techniques of optimization, game theory, and machine learning to cutting-edge applications such as cloud and mobile-edge computing, datacenters, resource allocation for 5G networks, and Internet of Things. He received the Best KHU Thesis Award in engineering in 2011 and several best paper awards, including IEEE ICC 2016, APNOMS 2016, and IEEE ICCS 2016. He receives the Korea NRF Funding for Basic Science and Research from 2016 to 2023. He has been the Editor of the IEEE TRANSACTIONS ON GREEN COMMUNICATIONS AND NETWORKING since 2016.

THANT ZIN OO received the B.S. degree in computing and information system from London Metropolitan University, U.K., in 2008, the B.Eng. degree in electrical systems and electronics from Myanmar Maritime University, Thanlyin, Myanmar, in 2008, and the Ph.D. degree in computer science and engineering from Kyung Hee University, South Korea, in 2017. He received scholarships from Kyung Hee University for his graduate studies and from British Council for his undergraduate studies. His research interests include wireless communications, network virtualization, data centers, sustainable energy, optimization techniques, and artificial intelligence.

CHOONG SEON HONG (S'95–M'97–SM'11) received the B.S. and M.S. degrees in electronic engineering from Kyung Hee University, Seoul, South Korea, in 1983 and 1985, respectively, and the Ph.D. degree from Keio University in 1997. In 1988, he joined KT as a Member of Technical Staff, where he was involved in broadband networks. In 1993, he joined Keio University, Japan. He was with the Telecommunications Network Laboratory, KT, as a Senior Member of Technical Staff and as the Director of the Networking Research Team until 1999. Since 1999, he has been a Professor with the Department of Computer Science and Engineering, Kyung Hee University. His research interests include future Internet, ad hoc networks, network management, and network security. He is a member of the ACM, the IEICE, the IPSJ, the KIISE, the KICS, the KIPS, and the OSIA. He has served as the General Chair, the TPC Chair/Member, or an Organizing Committee Member for international conferences, such as NOMS, IM, APNOMS, E2EMON, CCNC, ADSN, ICPP, DIM, WISA, BcN, TINA, SAINT, and ICOIN. He is currently an Associate Editor of the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, *International Journal of Network Management*, and the IEEE JOURNAL OF COMMUNICATIONS AND NETWORKS, and an Associate Technical Editor of the *IEEE Communications Magazine*.