Manas: Mining Software Repositories to Assist AutoML

Giang Nguyen
nguyen@iastate.edu
Dept. of Computer Science, Iowa State University
Ames, IA, USA

Rangeet Pan
rangeet@iastate.edu
Dept. of Computer Science, Iowa State University
Ames, IA, USA

Md Johirul Islam
mislam@iastate.edu
Dept. of Computer Science, Iowa State University
Ames, IA, USA

Hridesh Rajan
hridesh@iastate.edu
Dept. of Computer Science, Iowa State University
Ames, IA, USA

ABSTRACT

Today deep learning is widely used for building software. A software engineering problem with deep learning is that finding an appropriate convolutional neural network (CNN) model for the task can be a challenge for developers. Recent work on AutoML, more precisely neural architecture search (NAS), embodied by tools like Auto-Keras aims to solve this problem by essentially viewing it as a search problem where the starting point is a default CNN model, and mutation of this CNN model allows exploration of the space of CNN models to find a CNN model that will work best for the problem. These works have had significant success in producing high-accuracy CNN models. There are two problems, however. First, NAS can be very costly, often taking several hours to complete. Second, CNN models produced by NAS can be very complex that makes it harder to understand them and costlier to train them. We propose a novel approach for NAS, where instead of starting from a default CNN model, the initial model is selected from a repository of models extracted from GitHub. The intuition being that developers solving a similar problem may have developed a better starting point compared to the default model. We also analyze common layer patterns of CNN models in the wild to understand changes that the developers make to improve their models. Our approach uses commonly occurring changes as mutation operators in NAS. We have extended Auto-Keras to implement our approach. Our evaluation using 8 top voted problems from Kaggle for tasks including image classification and image regression shows that given the same search time, without loss of accuracy, Manas produces models with 42.9% to 99.6% fewer number of parameters than Auto-Keras’ models. Benchmarked on GPU, Manas’ models train 30.3% to 641.6% faster than Auto-Keras’ models.

CCS CONCEPTS

• Software and its engineering → Search-based software engineering.
• Computing methodologies → Machine learning.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

ICSE ’22, May 21–29, 2022, Pittsburgh, PA, USA
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9221-1/22/05.
https://doi.org/10.1145/3510003.3510052

1 INTRODUCTION

An increasingly larger number of software systems today are including deep learning. Deep learning uses a convolutional neural network (CNN) model, essentially a graph with weighted edges and nodes that are weighted functions, to convert inputs to the output. As more software systems incorporate deep learning, more software developers have to design and train CNN models. Designing a CNN model is very difficult, and developers often struggle leading to bugs. Model bugs are frequent bugs in CNN programs [29, 30].

Recent work on neural architecture search (NAS) aims to solve this problem [61]. NAS techniques start from a collection of default CNN models and search for a suitable model for the problem. The search space is defined by the collection of default models and a collection of mutation operators that are used to modify CNN models to create new candidates. NAS techniques have been implemented in industrial-strength tools such as Auto-Keras [31]. NAS techniques face two problems. First, NAS can be very costly, e.g., Auto-Keras takes 8-12 hours on high performance machines to search for models with reasonable accuracy (90+%). The accuracy drops rapidly if the search time is reduced. Second, CNN models produced by NAS can be very complex that makes it harder to understand them for maintenance, and costlier to train and retrain them.

We introduce Manas (Mining Assisted Neural Architecture Search) to alleviate the limitations of NAS. The fundamental intuition behind Manas is that mining and using the hand-developed models that are available in open-source repositories as default models or starting point of search can help NAS leverage human developer efforts. Manas applies this intuition in two ways. First, hand-developed models are mined to search for a better starting point for NAS. Second, the change patterns of the hand-developed models are mined to identify more suitable mutation operators for NAS.

We have realized Manas by extending Auto-Keras, the state-of-the-art NAS framework. Auto-Keras is open source and outperforms state-of-the-art methods like SEAS [17], NASBOT [32] making it a suitable baseline [31]. Some key technical contributions in Manas...
include model matching, a technique for matching the problem that the developer intends to solve with the hand-developed models mined from repositories, model adaptation, a technique for adapting the mined model to the problem context, model transformation, a technique for adapting the mined model for further improving metric values, and training adaptation that leverages mined parameter values from the repositories to change the optimizer.

To evaluate Manas, we use the top-8 problems from diverse domains obtained from Kaggle for machine learning tasks including image classification and image regression. Our evaluation shows that given the same amount of searching time, Manas generates simpler neural architectures than Auto-Keras without losing accuracy. In terms of the models’ size, Manas’ models have 42.9% to 99.6% fewer numbers of parameters than Auto-Keras’ models. We observed up to 641.6% faster training speed when training models produced by Manas as compared to those produced by Auto-Keras.

Our main contributions are the following:

- We have proposed a novel approach for NAS that leverages software repository mining.
- We have proposed methods to identify the suitable models by analyzing data characteristics and adapting models.
- We have utilized the common patterns to transform mined models to improve the performance of these models in terms of error rate, MSE, model complexity, and training time.
- We have implemented these ideas in a SOTA NAS framework, Auto-Keras [31]. Our artifact is available here [21].

The paper is organized as follows: §2 presents a motivating example, §3 presents preliminaries and problem statement, §4 describes the Manas approach for NAS, §6 describes the limitations and threats to validity of Manas, §7 describes related work, and §8 concludes.

2 MOTIVATION

Deep Learning has received much attention in both academia and industry. Therefore, many deep learning libraries and tools are created for supporting a large number of deep learning developers. Although these libraries and tools make deep learning more accessible, there are still many challenges. One of the challenging problems is constructing an appropriate CNN model architecture [53, 54], which also has been shown as a frequent bug in CNN programs by Islam et al. [29, 30]. For instance, Figure 1a shows a query [1] posted on StackOverflow where a developer is unable to find an appropriate CNN model for their purpose. In particular, the question discusses the difficulty that the ResNet architecture does not give the result as the developer expected. In response, an expert suggests changing the CNN model. Figure 1b shows the solution of the expert for the question of the developer. The expert suggested that the developer should add dropout layers to minimize overfitting.

Neural architecture search (NAS) aims to solve this problem [61]. NAS takes the training data as an input to automatically define the neural network for that data. Moreover, NAS is able to tune the hyperparameter of the searched neural network. There are both commercial and open-source realizations of NAS. For example, a developer can pay about $20 per hour to use Google’s AutoML. Auto-Keras is an AutoML system using NAS [31] created as an open-source alternative. Auto-Keras returns outstanding results compared with state-of-the-art handmade models on CIFAR10 [33], MNIST [34], and FASHION [57] datasets. Auto-Keras is shown to outperform state-of-the-art methods like SEAS [17], NASBOT [32].

NAS has two limitations. First, it can be very costly. For example, Auto-Keras consumes 2,300% more GPU computation time compared to using handmade model. Second, NAS often produces complex models that are hard to understand and time-consuming to train. To illustrate, we used Auto-Keras on another dataset, which is Blood Cell [39] collected from Kaggle. The model created by Auto-Keras for the Blood Cell problem has more than 2.3 million learnable parameters and more than 7 weight layers. The searched CNN models are constructed based on the architecture of the large default CNN models; thus, the models produced by Auto-Keras are often really large. Smaller CNN models train faster and save more energy [27, 44]. Han et al. have shown that reducing the number of parameters of deep learning models can reduce the training time by 3× to 4×, and energy consumption by 3× to 7× [24]. The rest of this work describes our approach Manas that addresses these limitations. As an example, for Blood Cell dataset, Manas produces a model that decreases the error rate by 47.1%, the model depth by 14.3%, the model width by 87.0%, and increases the training speed by 56.9% compared with Auto-Keras’s model. The model produced by Manas has 6 layers and 0.3 million parameters (neurons), whereas the model produced by Auto-Keras has 7 layers, 2.3 million parameters.

![Figure 1: An example from StackOverflow showing the necessity to change model architecture](image-url)
3 PRELIMINARIES AND PROBLEM STATEMENT

3.1 Preliminaries (NAS)
We define NAS like it was defined in Auto-Keras [31]. Given a search space \( S \) and the input data \( D \) split into \( D_{\text{train}} \) and \( D_{\text{val}} \), the goal is to find an optimal neural architecture \( N \in S \), which achieves the lowest value of the cost function in dataset \( D \). Search space \( S \) covers all the neural architectures created from default neural networks.

\[
N = \arg\min_{N \in S} \text{Cost}(N', D_{\text{val}}) \quad (1)
\]

\[
\theta' = \arg\min_{\theta} \mathcal{L}(N'(\theta), D_{\text{train}}) \quad (2)
\]

Where \( \text{Cost} \) and \( \mathcal{L} \) are the cost function and loss function, \( \theta \) is the learned parameter of initial architectures \( N' \).

3.2 Problem Statement
This work aims to utilize the neural networks from open source repositories to optimize the neural architecture search. We extract the data characteristics from both the input dataset and mined neural networks to determine better starting points (initial models) for NAS. Instead of using concrete default neural networks, the goal is to find optimal initial architecture \( N^* \) for NAS for each different input dataset. Similar to Equation 2, we obtain learned parameter \( \theta^* \) of new initial architectures \( N^* \):

\[
\theta^* = \arg\min_{\theta} \mathcal{L}(N^*(\theta), D_{\text{train}}) \quad (3)
\]

The optimal initial architectures \( N^* \) support NAS to find out the optimal neural network faster. In other words, with the same amount of searching time and input dataset, new initial architectures help NAS to find out a neural network with lower error compared to the original NAS:

\[
\arg\min_{N} \text{Cost}(N^*(\theta^*), D_{\text{val}}) < \arg\min_{N'} \text{Cost}(N'(\theta'), D_{\text{val}}) \quad (4)
\]

4 MANAS
Figure 2 shows the overview of Manas. Manas has five major components that we describe below.

1. To initialize Manas for NAS, the model database must be populated by mining models from open source repositories. Currently, Manas collects high quality models from GitHub by extracting Python files from Keras projects. These projects are selected using certain filtering criteria to ensure code quality. Then, API usage is used to filter Python files to those that contain models. Finally, both the models and the values for optimizer are extracted to store in the model database. This database is constructed once and should be updated periodically as new models are added frequently.

2. The data characteristics extracted from both the input data and mined models are used to select the suitable initial models for NAS from the database.

3. Model matching matches the data characteristics extracted from an input dataset and the mined models to obtain a good starting point. It selects the models, which have the closest data characteristics with the input dataset by using the model clustering approach. In case there are too many initial models, the model filtering approach will be applied to reduce the number of models.

4. The selected models are transformed by the model transformation approach based on related state-of-the-art papers and common layer API patterns of mined models. The transformation can enhance the performance of the models in terms of errors and training speed.

4.1 Model Mining
In order to extract CNN models and their optimizers from source code repositories, we build a source code analyzer based on the control flow graph (CFG). Figure 3 shows the overview of model mining process.
4.1.1 Meta-data collection. We collect the meta-data of GitHub repositories by using a GitHub API query. Meta-data contains basic information of a repository such as authors, repository’s name, etc. The query allows us to obtain the meta-data of the GitHub repositories with three filtering criteria including Python programming language, containing Keras keyword, and created date between 2015-01-01 and 2020-12-31. From the meta-data, we obtain GitHub URLs of the top 10,000 repositories with the most star count to ensure the quality of the models. The URLs help us to access the repositories to collect the CNN models.

4.1.2 Keras programs collection. We obtain 202,174 Python programs from collected repositories; however, only 42,139 programs use Keras API. In particular, we use CFG to analyze their import statements of Python programs to only collect which one imports Keras API.

4.1.3 Models extraction. In this work, since Manas only works with Keras CNN, we will explain how we extract CNN from Keras programs. We use CFG to extract a model from a deep learning program. We manually create a list of function calls used to build neural networks of Keras based on Keras’ documentation. Then, CFG examines every API call to collect the functions contained in the list and their connections. The collected functions represent the layers in the models. The connections between functions represent the layer connections in the models. The reason for using CFG is to collect complete models from programs even if they contain branches. For example, the CFG contains a convolutional block followed by a dense block in the “if” branch and a skip connection in the “else” branch. If a convolutional block combines with a dense block or a convolutional block combines with a skip connection to be a complete model, we will extract those parts in the branches to collect the complete model. CFG is used to handle the situation that there is a loop containing a part of a model in the CNN programs. The number of iterations is available to extract that part completely, we will collect it to complete the model. If there is a method call in the program containing a part of the model, we will collect it to complete the model. For the other cases, when the parts in the branches cannot complete the model or cannot be extracted, we will ignore them. For instance, a part of the model in a loop whose numbers of iterations are unavailable cannot be extracted. After this step, we collected 38,808 models from 42,139 Keras programs. The number of models is less than the number of Keras programs because many programs do not contain models. We assume that a program contains a model if it has at least an API used to build a model.

Rather than mining only neural architectures, we also mine their optimizers that deep learning developers carefully select after spending manual efforts on retraining their models. Then, whenever a model is selected as an initial model, Manas trains the model with its optimizer instead of the default one. Optimizers are algorithms deciding how the parameters of the models change. Every optimizer has strengths and weaknesses; thus, it is necessary to choose a suitable optimizer. While models are the decisive factor to the performance of Manas, we note that it is wasteful if we cannot fully use these models. In other words, good models with wrong optimizers cannot produce a good performance. To obtain the optimizer, we use the same process of extracting CNN models by creating a list of functions related to the optimizer. After that, CFG analysis is used to obtain the API call, which contains the optimizer.

4.1.4 Incomplete models detection. A complete model includes an input layer, hidden layers, and an output layer. The input layer is the first convolutional layer of the neural network, which is distinguished from the other convolutional layers based on its parameters. In particular, only the API call representing the input layer contains input_shape parameter. The output layer is the last linear layer of the CNN. Hidden layers are the layers between the input layer and the output layer, including convolutional layers, activation layers, and fully connected layers (linear layers). Therefore, if an extracted model does not have convolutional layers, activation layers, or linear layers, we will consider that the model is incomplete. By removing incomplete models, there are 29,846 models left.

4.1.5 Supported by Manas. As Manas currently supports a few kinds of layers that are the convolutional layer, the linear layer, the batch normalization layer, the concatenate layer, the add layer, the max pooling layer, the dropout layer, the Softmax layer, the ReLU layer, the flatten layer, and the global pooling layer, we filter out the models containing unsupported layers. After this step, we have 1,370 models left.

4.1.6 Model duplication detection. We obtain 793 models after removing the duplicate ones. We consider if two models have the same abstract neural network (ANN), there will be a duplicate model. We store the extracted CNN models in a database as an abstract neural network, which is an abstract representation of the neural network. This representation has the structure as a network where the nodes are API calls, and the connections are the order between API calls. We use this representation to adapt models and their optimizers into Manas. From each node, we obtain the name of layer and its parameters, which can be converted into an API call. Figure 4 presents an example of ANN built from a mined model. Notice that if an activation parameter is implemented as an argument inside a layer, we consider that the activation is a separate layer.

Figure 3: Model mining process.

\footnote{\url{https://api.github.com/search/repositories?q=keras+language:python+created:
 yyyy-mm-dd}}
Figure 4: Building an abstraction of neural network from extracted model

4.2 Data Characteristics Derivation

We analyze the input data and input/output layers of the mined models to extract their data characteristics. Taking an image classification as an example, we obtain the input data characteristics including the input size and the output channel from the image data. The data characteristics extracted from a mined model also are the input size and the output channel are obtained by analyzing the first convolutional layer and the last linear layer of a CNN model, which are the input layer and the output layer the model, respectively.

Example 1. Line 1 is an API call representing for the input layer of a CNN. We extract the value of the argument `input_shape` from the input layer to obtain the input size and the input channel, which are (120, 180) and 3, respectively. The output channel is obtained from the output layer 3. By extracting the first argument’s value of the output layer, we obtain the value of the output channel.

```
Conv2D(input_shape=[120, 180], activation='relu')
Dense(3, activation='relu')
```

Since we currently focus on image problems like image classification and image regression, we use the input size, the input channel, and the output channel of image as data characteristics. First, the input size includes the height and the width extracted from the input data and mined models. Second, the input channel represents the number of primary colors in the image. Third, the output channel is the number of output categories of the data and models.

4.3 Model Matching

Model matching is a ranking system used to find good models for a certain problem by using the data characteristics. Instead of using constant default models, model matching finds the suitable models to use them as new default models for NAS.

4.3.1 Model clustering. Model clustering uses the data characteristics of both the input dataset and mined neural networks to select appropriate initial architectures \(N^*\) for NAS. First of all, \(Manas\) clusters the mined models based on the data characteristics of the models and the input dataset. Secondly, in a meta-feature (data characteristics) space, our approach identifies closest clusters to the input dataset. Lastly, \(Manas\) uses all the models in the closest cluster to the dataset as the initial architectures for NAS. Formally, we determine the initial architectures \(N^*\) for dataset \(D\) in Equation 3 as follows:

\[
N^* = \{C_k \mid n \in C_k, C_k \in C\} \quad (5)
\]

\[
C = \{G(\Delta o_k, \Delta s_k) \mid \Delta i_k = 0, k \in [1, |S|]\} \quad (6)
\]

\[
dist(n, D) = \begin{cases} \min \Delta o \text{ if } \min \Delta i = 0 \\ \min \Delta s \text{ if } \min \Delta i = 0, \min \Delta o = 0 \end{cases} \quad (7)
\]

In Equation 6, \(C\) is a set of clusters of neural network detected by clustering algorithm G-means [23], which uses a statistical test to automatically decide the number of clusters. \(\Delta o_k, \Delta i_k, \Delta s_k\) and \(\Delta i_k\) are measured as follows:

\[
\Delta i_k = |i - i_k| \quad (8)
\]

\[
\Delta o_k = |o - o_k| \quad (9)
\]

\[
\Delta s_k = \sqrt{(w - w_k)^2 + (h - h_k)^2} \quad (10)
\]

Where \(i, o, w, h\) are the input channel, the number of output classes, the input width, and the input height of the dataset, respectively. Similarly, \(i_k, o_k, w_k, h_k\) are the input channel, the number of output classes, the input width, and the input height of a model \(k\), respectively. The idea behind the clustering equations is that there are two types of the input channel, including 1 and 3. Therefore, we classify the neural networks that have the same input channel first. Then, we use \((\Delta o, \Delta s)\) as the input of G-means to split the mined models into clusters. After that we identify the closest model \(n\) to the input dataset like Equation 7. The closest model is identified based on the priority of \(\Delta o\) and \(\Delta s\) that \(\Delta o\) takes precedence over \(\Delta s\). We have tried to run our tool in different orders of priority \(\Delta o\) and \(\Delta s\); however, this order of priority gives us the best results. The closest cluster to the input dataset is the cluster which contains the closest model. Then, we select all the models in the closest cluster to the input dataset as the initial models \(N^*\) for NAS shown in Equation 5.

4.3.2 Model filtering. If the number of initial models found by the model clustering approach is too large, we use model filtering to filter some models to increase the performance of \(Manas\). In the searching process, \(Manas\) trains all the default neural networks to select the best one. After that NAS is applied to tune the selected model. Thus, with a specific time budget, the more time \(Manas\) spends on trying the default models, the less time it spends on NAS for model searching. To filtering neural networks, we detect the equivalent architectures. We treat each neural architecture as a graph, whose trainable layers like convolutional layers or dense layers represent vertices, connections between two trainable layers represent edges, and channels of the outgoing trainable layers represent the weights of edges between two vertices. We use Cosine similarity to measure the similarity between two vertices. \(M_{ij}\) is an element of the similarity matrix of vertex \(v_i\) of a graph \(G_A\) and vertex \(v_j\) of a graph \(G_B\), which is measured as follows:

\[
M_{ij} = \frac{\sum_{k=1}^{n} w_{ik} w_{jk}}{\sqrt{\sum_{l=1}^{n} w_{il}^2} \sqrt{\sum_{l=1}^{n} w_{jl}^2}} \quad (11)
\]

Suppose that \(a_k\) is the common neighbor of vertex \(v_i\) and vertex \(v_j\), we have \(w_{ik}\) and \(w_{jk}\) are the weights of edges \(v_i a_k\) and \(v_k a_j\),
respectively. We also have \( w_{ij} \) or \( w_{ji} \) are the weights of edges that incident with each of the vertices \( v_i \) and \( v_j \), respectively.

\[
w_{pq} = \begin{cases} \text{weight of the edge between } v_p \text{ and } v_q \text{ in a graph } G \\ 0 \text{ otherwise} \end{cases}
\]

(12)

We create a similarity matrix \( M \) for each pair of graphs and classify equal matrices into the same classes. In this work, we filter the models by only using the neural network whose graphs belong to the class having the highest number of matrices. In other words, we will only choose the most used architecture in the determined initial architecture for NAS. We consider the most frequent architecture to be more suitable than others because many deep learning developers repeatedly use these neural architectures. Non-trainable layers like dropout or activation are not used as the factors to detect the similarities of the neural architectures because we want to utilize the various usages of these layers to increase the performance of Manas. Particularly, different models can have the same graph structure because of the differences in non-trainable layers; therefore, a graph can refer to many different neural architectures.

### 4.4 Model Transformation

Even though the mined models are from the top star GitHub repositories, they are not perfect. Therefore, we transform the initial architectures by modifying or adding the batch normalization (BN) layer, the flatten layer, the activation layers, the global average pooling (GAP) layer, and the dropout layer to optimize NAS in terms of speed, errors, and the number of parameters. We choose these layers to modify the default models because of two reasons. Firstly, we have found many common patterns related to these layers. Secondly, many recent studies have shown the effectiveness of these layers on increasing the performance of deep learning models. We have created a set of pre-defined rules to transform networks based on related state-of-the-art papers and common function calls’ patterns from mined models. Manas analyzes these architectures to determine whether the network satisfies the pre-defined rules. If these rules are satisfied, the pre-defined model transformations will be applied. The pre-defined model transformations support NAS to ignore transformations offered by our approach and focus on the other transformations. The model transformations do not work for Auto-Keras’ initial neural architectures because those default models have already included these transformations.

#### 4.4.1 Batch normalization layer constraint

We add a new batch normalization layer between the convolutional layer and the activation layer to increase training speed [28]. Using BN means that we modify the activations to normalize the input layer to decrease the training time. Many well-known neural architectures like ResNet [25], DenseNet [26], EfficientNet [50] use BN to increase the training speed. Many well-known deep learning developers repeatedly use these neural architectures. Non-trainable layers like dropout or activation are not used as the factors to detect the similarities of the neural architectures because we want to utilize the various usages of these layers to increase the performance of Manas. Particularly, different models can have the same graph structure because of the differences in non-trainable layers; therefore, a graph can refer to many different neural architectures.

#### 4.4.2 Global average pooling layer constraint

We use GAP to reshape the data into the correct format for fully connected layers to prevent overfitting [35]. Moreover, we use the mined models as default models for NAS; thus, the original input size of the initial models and the input size of the dataset can be different, which can cause a shape mismatch bug. However, using GAP can solve this problem since it does not care about the input shape.

**Example 3.** According to Figure 5a, the original model used the flatten layer to pass the feature map through the CNN. Therefore, in Figure 5b, following the constraint about GAP layer, Manas transforms flatten into GAP like the following example.

### 4.4.3 Activation layer constraint

We investigate the patterns of usages of activation functions used in the mined model. We have found 3218 hidden layers used in 793 models, where ReLU is used 2946 times, accounting for 94.55%. Therefore, we replace the current activation layers of convolutional layers with ReLU.

**Example 4.** According to Figure 5a, the original model uses Tanh for the convolutional layer. Thus, following the constraint of the activation layer, Manas transforms Tanh to ReLU like following example like the following example.
We implement Manas: Mining Software Repositories to Assist AutoML ICSE ’22, May 21–29, 2022, Pittsburgh, PA, USA

We evaluate the efficiency and effectiveness of model transformation whether using dropout is good or bad. The algorithm NAS used by Manas include model matching, model transformation, and NAS. The algorithm NAS used by Manas explores the search space via morphing the neural architectures guided by the Bayesian optimization algorithm. Mining the models is the most time-consuming task, which took about 24 hours to complete mining all the models. However, we only do this one time, so we do not count the time to mine the model in the comparison of Manas and Auto-Keras.

5.2 Mined Models
Since many models for different Kaggle datasets have already been published on GitHub, it is possible that some Kaggle models of the testing dataset have already been included in our mined models. To avoid this problem, for each dataset, we have examined all GitHub repositories of selected initial models of each dataset. There is no information showing that those initial models are specifically created for the input dataset. Moreover, for each input dataset, we have compared the initial model of Manas with all models from Kaggle. Manas’ initial model is not one of Kaggle’s models. Moreover, since most of the models from Kaggle are published as Jupyter Notebooks, we only mine the model written as Python files.

We only collect models from the top 10,000 repositories with the most star count to ensure high-quality models. We also do sanity checks by removing incomplete models and duplicate models. Moreover, we evaluate the mined models by training them on MNIST with 50 epochs [44]. The average accuracy of the mined models on MNIST is 90.98%.

5.3 Datasets
To evaluate the performance of our method, we use 8 different image datasets collected from Kaggle: Blood Cell [39], Breast Cancer [40], Flower [37], Intel Image Classification (IIC) [5], Malaria [3], MNIST: Ham [36], Sign Language Digits (SD) [38], and Sign Language (SL) [51]. Our goal was to utilize more complex datasets compared to well-known datasets such as MNIST, CIFAR10, and FASHION. Most of our datasets have large sizes of images. For example, in the intel image classification [5] dataset, the image size is 150x150 while the image size of MNIST is only 28x28. Secondly, the number of images in our datasets is much larger than MNIST, CIFAR10, and FASHION. For instance, the breast cancer dataset [40] has 277,524 images. Lastly, the number of classification of the evaluated datasets is up to 10 classes. These dataset are often used for image classification task; however, we treat prediction targets as numerical values for image regression task. For example, we will treat the prediction targets of the MNIST dataset as integers ranging from 0 to 9 as numerical values to be directly used as the regression targets. We only use image datasets because Auto-Keras only apply NAS for image data. We divide it into two subsets, 80% of randomly selected images are used for training and the remaining images for validation.

5.4 Results
5.4.1 RQ1: How efficient is Manas? To evaluate the efficiency of Manas, we run both Manas and Auto-Keras on 8 datasets for image classification and image regression. We vary the search time from 2 hours to 20 hours, which are described in Figure 10. Table 1 shows the error rate, MSE, the depth, the number of parameters, and the training speed of the best models of Manas and Auto-Keras. By the best model, we mean one that has the lowest error rate or MSE after search timeout. We run both Manas and Auto-Keras 5 times for each dataset with random training and validation sets. In the table, we report the information of the lowest errors of Manas and Auto-Keras [15]. By comparing Manas and Auto-Keras, two conclusions can be drawn.
First of all, Table 1 shows that Manas produces models, which has lower error rate or MSE compared with Auto-Keras’ models. The lower errors of Manas compared with Auto-Keras indicates that using mined models as the starting points for NAS can produce better models. Notably, Manas outperforms Auto-Keras by achieving 17.6% lower error rate on average. For some problems like IIC and MNIST: HAM for image classification task, the decrease of the error rate of Manas compared with Auto-Keras is small because the models created by Manas and Auto-Keras have reached the limit of error rate. Therefore, a small decrease in error rate is a major improvement. We use the error rate as the main evaluation metric to clearly point out this improvement of Manas to Auto-Keras.

Secondly, Manas achieves lower errors with less complicated models compared to Auto-Keras. Using mined models on NAS significantly decreases the complexity of the produced models. On average, models generated by Manas are 75.7% less deep and 93.0% less wide compared to the models generated by Auto-Keras for image classification task. Similarly, Manas creates models with 74.4% less deep and 89.1% less wide compared to Auto-Keras’ models for image regression task. Simpler DNN models train faster and save more energy [27] than the complex ones. Han et al. have shown that reducing the number of parameters of deep learning models can reduce the training time by 3x to 4x, and energy consumption by 3x to 7x [24]. Table 1 also shows that on average, Manas’ models run faster than Auto-Keras’ model 247.1% and 66.3% in image classification and image regression, respectively.

5.4.2 RQ2: How efficient are model transformation and optimizers modification? We create an ablation study to observe the efficiency of the model transformation and modifying optimizers via mining separately.

- **Original Manas** (OM) represents mined models + no transformation + no optimizer + NAS.
- **Transformed Manas** (TM) represents mined models + transformations + no optimizer + NAS.
- **Manas** (MN) represents for mined models + transformations + optimizers + NAS.
- **Auto-Keras** (AK) represents NAS.

We observe the error values of Original Manas, Transformed Manas, Manas, and Auto-Keras by executing them on 8 datasets for 20 hours for both image classification and image regression. To evaluate the efficiency of the model transformation, we compare Original Manas and Transformed Manas. We compare Transformed Manas and Manas to evaluate the efficiency of optimizers modification. To evaluate the combination of all methods, we compare Manas and Auto-Keras.

Notice that we have applied the model transformation on Auto-Keras’ models, however, this did not lead to any improvement because those default models use BN layers or Dropout layers appropriately. Figure 10b does not have the Original Manas series because the best models of Original Manas and Transformed Manas are the same. In other words, the model already contains all the transformation constraints, so there is no transformation applied to the model. Similarly, Figure 10e does not have the Transformed Manas series because the optimizers of the best model of Transformed Manas and Manas are the same. The selected model’s optimizer of this problem is not available; therefore, we use the default optimizer of Auto-Keras for this model. From the results are shown in Figure 10, we draw four observations.

First of all, model transformation increases the performance of Manas in terms of errors and converge time. Figures 10a, 10c, 10f, 10g show that Transformed Manas achieve lower errors than Original Manas most of the time. The activation layers, the GAP layer, and the dropout layers contribute to the better errors of Transformed Manas compared to Original Manas. For example, the dropout layers can prevent overfitting, which decreases the errors. The model transformation also helps Manas to converge faster. As we can observe Figures 10a, 10c, 10f, 10g, Original Manas has higher errors compared to Transformed Manas in the first 2 hours of training, which indicate the Transformed Manas converge faster than Original Manas. Transformed Manas has a fast converge speed thanks to BN.

Secondly, mined optimizers help Manas to reduce errors and converge faster. In most of problems, the best models of Manas have lower errors than the best models of Transformed Manas. Moreover, Manas can converge easier with the mined optimizers. For instance, the Figures 10d, 10f, 10g shows that Transformed Manas has trouble converging. After 8 hours of searching, Transformed Manas cannot find out better models while Manas succeeds. Transformed Manas has trouble in converging since it does not have an suitable optimizer for those problems like Manas.

Thirdly, Auto-Keras may have lower errors than Manas in the first few hours; however, in the last hours, Manas often finds out better models than Auto-Keras. For example, Figures 10b, 10c, 10d, and 10m show that Auto-Keras has the errors at the beginning; however,
as time gone we have not seen any improvement in Auto-Keras results. The reason for this problem is that Auto-Keras starts with very complicated models that take a lot of time to train; therefore, the number of models is searched by Auto-Keras are small, which decreases the chance to find out good models of Auto-Keras. Manas starts with simple models, which trains faster. Therefore, Manas may not be off to a good start, but in the end, it still outperforms Auto-Keras. The observation shows the benefit of using simple mined models in NAS.

Lastly, Figure 10 indicates that Manas obtains lower error rates than Auto-Keras in almost different time periods. As we can observe in Figures 10a and 10i, Manas always outperforms Auto-Keras in terms of error rate in the 20-hours of searching. Figure 10 shows that the longer searching time sometimes gives worse models. During the searching process, the NAS estimates the errors of searched models and selects the best one. However, the estimation may not be accurate, leading to an incorrect choice. This problem of NAS indicates the importance of using a simple model as a starting point. Simple models can train faster that increases the chance for NAS
to search for more models. Since the estimation of NAS can be incorrect; thus, if we increase the number of searched model, we can increase the chance to obtain better models. For example, after 8-hours of searching, NAS goes wrong with IIC problem when it produces worse models than before. However, when we keep searching for new models, NAS gradually fixes the problem to obtain a good model.

5.4.3 RQ3: How efficient is model matching? The goal of model matching is not only selecting good default models for Manas but also reducing the number of default models. From Table 1, we observe the efficiency of model matching, when Manas can outperform Auto-Keras in many different perspectives. Table 2 shows that model matching, including model clustering and model filtering, can significantly reduce the number of default models for Manas. To balance the time that Manas spends between initial architectures and NAS, we applied the models filtering when the number of initial neural networks is larger than 40 since Manas often takes more than 10 hours, half of the amount of time used in our evaluations to complete training these neural networks. We use all 793 mined models as input for each problem, which may take few days to complete training. By using model matching, we decrease the number of default models by 99% on average. Taking SD in image classification as an example, after using DNN clustering, there are remaining 69 default models. Therefore, we use DNN filtering on this dataset to reduce the number of models of SD from 69 models to 12 models, which eliminates 82.6% number of the DNN model.

5.4.4 RQ4: How efficient is Manas for well-known problems? We also evaluate Manas with the well-known datasets like FASHION, MNIST, or CIFAR10 for the image classification task. Table 3 shows the error rate, depth, number of parameters, and speed of the best models of Manas and Auto-Keras. As can be observed, Auto-Keras achieve better error rates on these datasets than Manas; however, Auto-Keras produces larger models to achieve these error rates while Manas uses much smaller models to get close to the error rates of Auto-Keras in CIFAR10 and MNIST problems. Particularly, Manas’ models are 62.3% shorter and 89.9% than Auto-Keras’ models on average, which increases the training speed of Manas’ models by 73.3% compared to Auto-Keras’ models. Auto-Keras achieves better error rates than Manas since Auto-Keras uses ResNet and DenseNet as initial models. These neural networks are well-tuned to achieve outstanding results on these datasets, which once again shows that using good initial architecture optimizes NAS.

### 6 LIMITATIONS AND THREATS TO VALIDITY

6.1 Limitations

In this work, Manas directly derives the data characteristics from the image dataset, which is limited to the image classification and image regression problems. We believe that Manas is not directly applied to other problems such as natural language processing (NLP) or video classification; however, our approach of mining models to identify a good starting point candidate should be applicable to any AutoML problems. For example, data characteristics of NLP problems also include the input shape, which are input time steps and the number of features, and the total number of output classes. Time steps represent the maximum length of the input sequence, which could either be the number of words or the number of characters depending on what we want. The number of features is the number of dimensions we feed at each time step. We can analyze the input layer and output layer to extract data characteristics from models. We can analyze the input dataset to obtain its data characteristics. The extracted data characteristics can be used to find better starting points for NLP problems in AutoML systems.

Manas can only work with few kinds of layers since it only uses the layers that Auto-Keras supports. This limitation can decrease the performance of Manas because if Auto-Keras supported more layers, we can mine more kinds of models from the software repositories.

6.2 Threats to Validity

6.2.1 Internal validity. We have tried our best to obtain the results of Manas and Auto-Keras on as many datasets as possible. Because of the time limit, Manas is currently evaluated on 8 datasets. All the source code, trained models, datasets, and evaluation data are public for reproduction to mitigate these threats.

6.2.2 External validity. First of all, Manas only focuses on image classification/regression problems. We rely on meta-features to find good starting points for NAS; therefore, one possible threat is that meta-features (data characteristics) do not work for other types of problems. However, Auto-Sklearn [20] and Auto-Sklearn 2.0 [19] mitigate this threat by showing that using meta-features can increase the performance of AutoML systems in terms of training speed and accuracy on structured datasets. Secondly, Manas only focuses on CNN. Thus, another threat is that the model transformation approach does not work for other types of models. Nevertheless, Cambronero et al. [10] propose AMS showing that using unspecified complementary and functionally related API components can improve the performance of AutoML systems for classical models such as Linear Regression or Random Forest. The difference between Manas and AMS is that AMS applies these transformations to search space while Manas applies these transformations to default models.
7 RELATED WORK

7.1 Neural Architecture Search

NAS is a technique for automatically finding appropriate neural architectures which can outperform most of the hand-designed neural networks. Specifically, NAS needs a training dataset as the input to create a powerful neural architecture. There are many different approaches for a NAS system; however, most of them have three main components which are search space, search strategy, and optimization strategy. The search space represents the search boundary of a NAS system limiting what kinds of neural network can be searched and optimized. For instance, Baker et al. [4] use the convolutional architecture with pooling, linear transformations as a search space. Around the same time, Zoph et al. [61] use a similar search space; however, the authors use more skip connection for the search space. The search strategy is used to search appropriate models in a defined search space. There are many approaches to search models such as reinforcement learning [4, 9, 59, 61, 62] or evolutionary algorithms [46, 49, 58]. This optimization strategy supports NAS to guide the network search process. The optimization strategy evaluates a searched model with training data without training these models. Recently, many methods are proposed to optimize NAS [32, 41, 45]. In our work, Manas mines the neural networks from repositories to enhance the power of NAS by supporting it to have a better starting point.

7.2 AutoML

AutoML is a process for constructing an appropriate model architecture for a specific problem automatically. Many features that AutoML can provide to deep learning users, such as automated data augmentation, automated hyperparameter tuning, or automated model selection. A lot of AutoML systems have been created like Auto-WEKA [52] on top of WEKA [22, 56], Auto-Sklearn [20] on top of Scikit-learn, which support deep learning user to automate tuning hyperparameter and model selection. Some other AutoML systems can support deep learning users to automate optimizing the full ML pipeline. For instance, TPOT [42, 43] uses evolutionary programming to optimize ML software. However, a main disadvantage of these systems are very slow because of the high GPU computation requirement. Recently, Auto-Keras is created to handle this problem, which has implemented network morphism [12, 55] to reduce the searching time of NAS. Network morphism is a technique to morph a neural architecture without changing its functionality. Nevertheless, even though Auto-Keras apply network morphism technique, it still takes a lot of GPU computation. Our approach uses DNN model mining and common layer patterns to enhance the performance of AutoML system.

7.3 Mining Software Repositories

Cambronero et al. [11] proposed AL, a system that leverages existing machine learning code from repositories to synthesize final pipelines. AL can generate ML pipelines for a wide range of problems without any manual selection. Cambronero et al. [10] also proposed AMS, which automated generates new search space for AutoML systems by utilizing source code repositories. The new search space is created based on an input ML pipeline, which increases the performance of AutoML systems. However, these only operate classical machine learning models, whereas Manas works with neural networks.

7.4 Meta-features

Auto-Sklearn [20] uses 38 meta-features of structured datasets to find a better starting point. Feurer et al. [19] proposes Auto-Sklearn 2.0, which reduces the number of meta-features to three, including the number of data points, the number of features, and the number of classes. The reasons for the reduction are that good meta-features are time-consuming and memory-consuming to generate. We also do not know which meta-features work best for which problem. Unlike Auto-Sklearn and Auto-Sklearn 2.0, Manas uses meta-features to find a better starting point for NAS, which works for neural networks. Moreover, Manas also proposes the meta-features, which helps NAS find a better starting point for image datasets.

8 CONCLUSION

We present Manas, a technique for NAS, which uses the mining technique to assist NAS. The key idea of Manas is to mine models from repositories to enhance NAS. In particular, we use CNN models mined from software repositories as the default model of NAS. From a large number of models, we use the model matching approach to find good models for a problem. We also apply some transformations for those models to enhance their performances. With better default models, Manas can increase NAS’s performance, which leads to better CNN models as search results. Our experiment shows that Manas can produce better CNN models in terms of the error rate and MSE, the model complexity, and the training speed than Auto-Keras. Future work will involve extending Manas to problems beyond those tackled in this paper, such as video classification. We can utilize the code change patterns [16] in ML programs to improve the results. Moreover, the proposed approach of Manas can be applied to other automated tools, e.g., AutoAugment [14] of other components in ML pipeline [7]. The technique can also be utilized to use AutoML to address the other problems in ML, such as fairness bug [6].

ACKNOWLEDGMENTS

This work was supported in part by US NSF grants CNS-21-20448, CCF-19-34884, and Facebook Probability and Programming Award (809725759507969). We also thank the reviewers for their insightful feedback for improving the paper.

REFERENCES

[1] Anonymized. 2015. Resnet network doesn’t work as expected. https://stackoverflow.com/questions/49226447/resnet-network-doesnt-work-as-expected
[2] Anonymized. 2021. Keras documentation. https://keras.io/api/
[3] Arunava. 2018. Malaria Cell Images Dataset. https://www.kaggle.com/iarunava/cell-images-for-detecting-malaria
[4] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. 2017. Designing Neural Network Architectures using Reinforcement Learning. In 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net. https://openreview.net/forum?id=Stc2vqpc
[5] Puneet Bansal. 2018. Intel Image Classification. https://www.kaggle.com/puneet6060/intel-image-classification
[6] Sunon Biswas and Hridhek Rajan. 2020. Do the Machine Learning Models on a Crowd Sourced Platform Exhibit Bias? An Empirical Study on Model Fairness.

[Manas: Mining Software Repositories to Assist AutoML ICSE ’22, May 21–29, 2022, Pittsburgh, PA, USA]
[51] Tecperson. 2017. Sign Language MNIST. https://www.kaggle.com/datamunge/sign-language-mnist
[52] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 847–855.
[53] Mohammad Wardat, Breno Dantas Cruz, Wei Le, and Hridesh Rajan. 2022. DeepDiagnosis: Automatically Diagnosing Faults and Recommending Actionable Fixes in Deep Learning Programs. In ICSE’22: The 44th International Conference on Software Engineering (Pittsburgh, PA, USA).
[54] Mohammad Wardat, Wei Le, and Hridesh Rajan. 2021. DeepLocalize: Fault Localization for Deep Neural Networks. In ICSE’21: The 43rd International Conference on Software Engineering (Virtual Conference).
[55] Tao Wei, Changhu Wang, Yong Rui, and Chang Wen Chen. 2016. Network morphism. In International Conference on Machine Learning. 564–572.
[56] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. 2016. Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann.
[57] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747 (2017).
[58] Lingxu Xie and Alan Yuille. 2017. Genetic cnn. In Proceedings of the IEEE International Conference on Computer Vision. 1379–1388.
[59] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. 2018. Practical block-wise neural network architecture generation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2425–2432.
[60] Runjie Zhu, Xinhui Tu, and Jimmy Xiangji Huang. 2020. Chapter seven - Deep learning on information retrieval and its applications. In Deep Learning for Data Analytics, Himansu Das, Chittaranjan Pradhan, and Nilanjana Dey (Eds.). Academic Press. 125–153. https://doi.org/10.1016/B978-0-12-819764-6.00008-9
[61] Barret Zoph and Quoc V. Le. 2017. Neural Architecture Search with Reinforcement Learning. In 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net. https://openreview.net/forum?id=r1Ue8Hcxg
[62] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning transferable architectures for scalable image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition. 8697–8710.