Visual Steering for One-Shot Deep Neural Network Synthesis

Anjul Tyagi, Cong Xie, Klaus Mueller

Fig. 1. Our visual steering interface purposed to guide analysts in the task of constructing the best performing deep neural network architecture for a given application using a one-shot search algorithm. The first section is the Lego View where the analyst can create and edit different components of a large neural network with simple drag and drop operations. The Lego View visualizes the different neural network components along with their parameters. An initial large neural network is treated as a super graph (shown in the Graph View) and the problem of finding the best performing neural network architecture is framed as searching for the respective subgraph in this super graph. The Graph View visualizes the super graph where each node is a block (sequence of neural network components). The One-Shot Search algorithm evaluates the subgraphs of this super graph iteratively, gauges their accuracy with regards to a test dataset and provides a fitness score for each node in the graph (Block Information view). The subgraphs are then projected as points into the scatterplot in the Search Space view and colored based on their evaluation accuracy. Analysts can filter and analyze a specific region in the subgraph search space with zoom and pan operations in the Search Space View. Finally, all blocks with high fitness scores are combined to create the best performing candidate neural network architecture.

Abstract—Recent advancements in the area of deep learning have shown the effectiveness of very large neural networks in several applications. However, as these deep neural networks continue to grow in size, it becomes more and more difficult to configure their many parameters to obtain good results. Presently, analysts must experiment with many different configurations and parameter settings, which is labor-intensive and time-consuming. On the other hand, the capacity of fully automated techniques for neural network architecture search is limited without the domain knowledge of human experts. To deal with the problem, we formulate the task of neural network architecture optimization as a graph space exploration, based on the one-shot architecture search technique. In this approach, a super-graph of all candidate architectures is trained in one-shot and the optimal neural network is identified as a sub-graph. In this paper, we present a framework that allows analysts to effectively build the solution sub-graph space and guide the network search by injecting their domain knowledge. Starting with the network architecture space composed of basic neural network components, analysts are empowered to effectively select the most promising components via our one-shot search scheme. Applying this technique in an iterative manner allows analysts to converge to the best performing neural network architecture for a given application. During the exploration, analysts can use their domain knowledge aided by cues provided from a scatterplot visualization of the search space to edit different components and guide the search for faster convergence. We designed our interface in collaboration with several deep learning researchers and its final effectiveness is evaluated with a user study and two case studies.

Index Terms—Deep Learning, Neural Network Architecture Search, Visual Analytics

1 INTRODUCTION

Neural Networks (NN) have been successfully applied in many applications, including Computer Vision [65], Natural Language Processing [69], Data Analytics [45], etc. With the recent advances in computing power of computers and big data, the size of the neural networks that could be successfully trained has increased by manifolds [20]. It leads to the inherent problem of tuning millions of associated parameters. The root of the complexity in NN architecture search is the training and evaluation of the candidates in a large search space. For instance, Google net has 22 layers and about 95 components (e.g., Convolution or Max Pooling). Suppose the components are grouped into blocks to test multiple combinations of components (see Figure 2 for an exam-
In such a scenario, the total possible NN architectures would be on the order of $2^{3.6} \approx 4.0\times 10^9$. Each solution in the search space would take a long time for training as there is a large number of parameters to be tuned (e.g., 6.8 million parameters for GoogleNet). All of these possible architectures constitute a huge search space, where even the most powerful computers will spend years of CPU time on finding a near-optimal solution. Furthermore, the tuning of each component itself is far more diverse, for example, there are several parameters like step, stride, kernel size, padding, and activation function associated with a regular convolution unit in a neural network.

Some well studied deep neural networks like AlexNet [29] and ResNet [20] have been the results of extensive architecture search studies and hours of manual parameter tuning by experts. Most of the current automated approaches find the optimal solution of a NN architecture based on adaptive experiments [50], and most of them rely on strong computing power, such as frameworks built on top of a whole cloud network [66]. An example of this can be seen in the work by Zoph et al. [73] and Real et al. [52] where the search algorithms took almost 2,500 GPU days to find the best-performing neural network architecture. As a result, these networks are very difficult to generalize because of the very high hardware equipment demands and associated costs. There also exist some visual analytics tools to include humans in the loop along with automated search techniques [12]. However, they still require pre-trained candidate models to start the search process. To the best of our knowledge, most of the automatic or visual analysis approaches fail to provide an efficient search algorithm. Training an abundance of candidate architectures is hence unavoidable.

One-shot search is one of the latest algorithms designed to reduce the high cost of training. It constructs a super-graph of the network architecture as shown in Figure 1, which is trained only once. Then the subgraphs of the graph are evaluated and selected as the candidates for architecture search. However, what is still missing is an efficient way of searching for the optimal subgraphs in the huge search space of subgraphs. Using human knowledge can improve the efficiency of this, for example, knowing that batch normalization is supposed to go after a convolution layer. One might think that knowledge of this kind can be easily encoded into an automated search scheme, but the arsenal of tricks of the trade is endless. The creative mind of human experts always has an edge, which lies at the heart of visual analytics. Hence, constructing a visual analytics approach that follows human experts filter such subspaces by applying their creative domain knowledge can guide this search to reach the optimum much faster.

In this work, we present a novel visual analytics interface, developed in close collaboration with deep learning researchers to solve the prevailing problem of deep neural network architecture search. Our technique is more effective than the fully automatic approaches, manual parameter tweaking approaches, and the existing visual analytics frameworks designed for similar tasks. A single large neural network with many repeated components (super-graph) constructed by combining several blocks (refer to Figure 2) is trained only once. We implemented an evolutionary search algorithm that then evaluates a small set of sub-networks from this trained neural network and gauges their accuracy on a validation dataset. These accuracies are further used to generate rankings for each block in the original trained large neural network.

With each iteration of the search algorithm, more and more sub-graphs are evaluated, hence generating more accurate fitness values for each block. A combination of the best-scored blocks is used to form a candidate neural network with the best performance. The major advantage of using the one-shot search algorithm is that it requires only a single large neural network to be trained, instead of many candidate neural networks, thus vastly reducing the number of parameters to be trained.

Besides providing an efficient evolutionary search algorithm, our visual analytics approach also solves two basic limitations with the automated approach of one-shot architecture search. First, it is hard to come up with pre-trained candidate neural network architectures to begin with. We tackle this problem by providing template networks for the majority of applications of deep learning, carefully designed by experts; yet, analysts are still free to add and remove components from this template network at their discretion.

Second, even though the one-shot search technique is significantly better than the other fully automatic architecture search algorithms, it still can be very slow and resource-intensive to evaluate all possible sub-graphs of the network. To solve this issue, our interface offers a search-space view along with a block information view (refer to Figure 3) to help the analysts in deciding where best to prune a large search space of subgraphs (the candidate architectures). This allows the one-shot search algorithm to converge to a solution much faster. The two views are updated based on the evaluation of each iteration of the evolutionary search, and as more and more subgraphs are evaluated, the search-space view allows the analysts to understand which region of the search space is giving the best candidate architectures.

Overall, the main contributions of our paper are as follows:

- a novel visual approach for the design and construction of the solution space of one-shot neural network search architecture
- an evolutionary search algorithm for candidate neural network architectures, supporting our human-guided search scheme
- an intuitive plug and play visual analytics framework to design and evaluate deep neural networks

Our paper is organized as follows. Section 2 presents related work. Section 3 explains the working of the one-shot neural network architecture search algorithm. Section 4 lists the tasks solved by our interface which are some of the common problems faced by the deep learning community related to the deep neural network architecture search. Section 5 describes the implementation and description of various components of the interface in detail. Section 6 outlines a thorough evaluation we performed with the help of a user study and two case studies to show the generality of the tool. Section 7 concludes.

2 RELATED WORK

2.1 Automatic Neural Network Architecture Search

Automated Neural Network Architecture search has a long history [11][55]. Moreover, after the recent research showing promising results with the application of deep neural networks in the areas of Computer Vision [15][29][32][33], Natural Language Processing [40][57][60], Storage Systems [10][63] and other applications [20][54], the problem of finding the best working neural network architecture has escalated. Neural Networks are constantly growing in size, for example, AlexNet [29] developed in the year 2012 for image classification task had 8 layers, which was followed by VGGNet [59] in the year 2014 which had 19 layers. Presently, these networks have become as big as containing hundreds of layers [20][61]. The number of hyperparameter tweaks required to make these deep neural networks work for a particular application can easily reach up to a few million. Several techniques have been applied to automatically compute the best performing set of hyperparameters in deep neural networks including Reinforcement learning [23][73], Bayesian Optimization [9][26], and Genetic Algorithms [16][41]. All these techniques are applied in the process called NAS (Neural Architecture Search) [50][68] designed to automate neural network architecture engineering [16].

NAS designed methods have outperformed manually curated networks as shown by Zoph et al. [50], Real et al. [52] and the SMASH...
model [9]. They use several trained networks to provide the final architecture design after evaluating each network on a validation set. However, training networks with NAS is expensive since many different networks have to be trained before evaluation. To overcome this, another technique called the MorphNet [19] uses a different approach where the final architecture design is decided directly as a subset of a single overcomplete trained neural network. An overcomplete network is a large neural network with repeated components for testing, as shown in Figure 2 where repeated components are placed inside different blocks for clarity. Following the work on MorphNets, a slightly different approach known as the One-Shot Architecture search [7] has evolved which involves searching for the best neural network architecture as a subset of a large trained network by ranking each component inside the neural network. Using the One-Shot method to train a neural network has the benefit that the network only needs to be trained once, which although overcomplete, yet it has a lesser number of parameters than training several different architectures independently [7].

### 2.2 One-Shot Deep Neural Network Architecture Search

As discussed in the previous section, one-shot deep neural network architecture search has an advantage over other techniques since only one large network is required to be trained, which results in a reduced hyperparameter search space. Also, since the main goal of training is only to rank the components (blocks inside a neural network, for example, a 3x3 Convolution block, Average Pooling block, etc.) refer to Figure 2 with respect to each other, the network can only be trained in a few epochs to get the gist of rankings, irrespective of the accuracy of the network on the training dataset. Using the benefits of the one-shot architecture search technique, our tool provides an easy-to-use interface to make the process of architecture search more effective.

### 2.3 Explainable Machine Learning

Explainable machine learning plays an important role when it comes to diagnosing the performance of a neural network, which in turn helps in designing the neural network architecture. Visualization has been used extensively in explainable machine learning [12, 54] to understand the decision-making process of neural networks. Some important work in this area include the use of saliency maps [53] and the evaluation of local components of a network using the gradient-based methods [51]. Many specialized techniques have been developed for Convolutional Neural Networks because of their wide application [70] explaining how each layer in a deep CNN evaluates an image. Similarly, specialized techniques exist for RNNs which have been widely adopted in the Natural Language Processing community where the visualization techniques display how LSTMs learn the semantics of text, thus assisting in effective designing of the LSTM models [42]. Combining all this research, open-source libraries have been developed to make these visualization techniques readily available to users [48].

The main reason for developing these techniques is to assist in the manual curation of deep neural networks. These techniques have been useful in designing some well known deep neural network architectures like ResNet [20] and AlexNet [29]. However, developing a neural network architecture using these specialized techniques requires several man-hours and generally only works for a specific application, hence they cannot be generalized to other applications and are not scalable. In addition, evaluating several of these different architectures requires expensive resources and large training time [55]. Also, from the perspective of a beginner analyst in Machine Learning, it is hard to design large neural networks keeping in mind all the specializations of different components from within a network. Hence automatic methods for neural network architecture search are preferred for applications that have not been extensively studied in the deep learning community. Our tool provides an easy to use interface to help the analyst decide an initial viable deep neural network architecture for a specific application which can further be analyzed via existing tools, for example, Lucid [48] etc. for further analysis and evaluation.

### 2.4 Interactive Neural Network Architecture Search

There has been significant research in the visualization community to make the process of neural network model selection and search more effective. Techniques exist to evaluate network architectures where the models are already known and they are required to be compared on the same validation dataset [11, 14, 56, 71]. On the other hand, visual analytics frameworks have been proposed to get the human in the loop for effectively applying machine learning to different scenarios. BEAMES [15] helps the users to iteratively find the best regression models for a given dataset. TreePOD [43] provides an interface to manage the trade-off between accuracy and interpretability of different existing machine learning models that would fit a particular dataset. REMAP [12] allows interactive convolutional neural network architecture search starting with a few pre-trained models, which are used to identify potential architectures towards better accuracy. Besides designing neural networks, there also exist tools that allow interactive design and filtering of clustering techniques [13, 31, 46, 53] and dimension reduction [6, 14, 25, 37, 47]. However, considering the problem that it is hard to find a good deep neural network architecture as compared to regression, clustering, and dimension reduction models, our tool focuses on this problem by evaluating different components of deep neural networks based on the specific application problem. Also, there is no assumption of existing pre-trained networks for evaluation.

### 3 One-Shot Neural Network Architecture Search

This section will discuss the technical details about the one-shot deep neural network architecture search algorithm [7]. The overall approach for one-shot architecture search consists of four steps, (1) Designing a template network with many repeated components, also known as an overcomplete network that allows representing the search space for the deep neural network architecture components, (2) Training of the template neural network, (3) Evaluating the trained template network and the final step including (4) Re-training of the selected network on the dataset from scratch followed by evaluation. These four steps are described further in the following text.

**Designing a Template Neural Network:** While designing a template network, the main factor to be considered is the coverage of the search space. Since the search space of candidate neural network architectures is the space of subgraphs of this template network, the analyst should carefully choose a different variety of blocks in the network to provide maximum coverage. A single layer in the template network can consist of many blocks, which can have a different sequence of components, as shown in Figure 3. These outputs from each block are then merged with a concatenation or a sum operation before feeding the result to the next layer. In such a case, it is necessary to match the output parameters of each block for proper training of the template neural network. For example, a 3x3 convolutional layer with a padding size of zero will have the same output size as a 5x5 convolutional layer with a padding size of one. This output size matching is easy to handle with our interface since it automatically fixes the output size while the analyst is designing the template network. For example, following up on the previous example, when the analyst keeps a 3x3 and a 5x5 convolutional blocks on the same layer, our interface automatically fixes the padding size to zero and one respectively to match the output sizes from these blocks.

**Training of the Template Neural Network:** This search method is called One-shot Search because only a single large neural network...
We devised an evolutionary search algorithm to search for the candidate weights associated with many blocks of the template network are zeroed. The number of candidate neural networks (subgraphs) in the population are equal to the number of blocks in the template network. A sample mask that represents a subgraph from the template NN which includes the blocks corresponding to set bits is generated from the probability density functions of the population, and the mask is also generated from the probability density functions of the population. This mask is compared to the masks of the father and mother models to generate a child mask. Note that both the father and mother models are chosen from the best performing candidates from the population, and the mask is also generated from the probability density functions of the population.

### 3.1 Evolutionary Search Algorithm

We devised an evolutionary search algorithm to search for the candidate NN architecture iteratively. At each iteration, our search algorithm outputs the fitness scores for each block of the template NN as well as performance scores of some of the evaluated subgraphs from the search space. Every iteration evaluates more and more subgraphs for their performance, giving a better approximation for the fitness values associated with each block. Finally, the analyst can use the blocks with high fitness values to construct the final candidate NN architecture. An overview of this process is shown in Figure 4 and details are provided in Algorithm 1.

#### 3.1.1 Selecting the candidate models

The number of candidate neural networks (subgraphs) in the population is heuristically set to the number of blocks in the template neural network. To select each subgraph from the super-graph of a template neural network, we generate a mask of zeros and ones with of size equal to the number of blocks in the template network. A sample mask for a 9 block template NN is shown in equation (1):

\[
\text{mask} = [1, 0, 0, 1, 0, 0, 1, 1]
\]

This mask represents a subgraph from the template NN which includes the blocks corresponding to set bits. We generate a candidate neural network from the mask by zeroing out the weights of the blocks excluded from the template network. This way, only the blocks with a corresponding set bit in the mask are activated. It is necessary for the subgraph to have at least one block from the input layer and the block from the output layer for correct evaluation.

#### 3.1.2 Calculating the fitness values

To calculate the fitness of a subgraph, we calculate its loss value on the evaluation set. The higher the loss values, the lower the fitness value. Following this method, we calculate the fitness of the individual candidate architectures as shown in Equation (2). Then an overall fitness value is assigned to each candidate architecture by dividing the fitness value by the sum of fitness values for every architecture.

\[
\text{individualFitness} = \frac{1}{\text{loss} + \epsilon}
\]

#### 3.1.3 Choose Parents

The choose parents procedure returns the father and the mother neural network randomly chosen from the list of population models, with a higher probability of choosing the models with higher fitness values. These models are sampled from the fitness probability distribution of the population.

#### 3.1.4 Cross-Over

To generate the child architectures from the father and the mother neural network models, a cross-over procedure is used, shown in Algorithm 2.

Firstly, a mask is generated with the procedure described in Section 3.1.1. This mask is compared to the masks of the father and mother models to generate a child mask. Note that both the father and mother models are chosen from the best performing candidates from the population, and the mask is also generated from the probability density of the fitness values, i.e., more probability of bits being set at indices pointing to well-performing blocks.

#### 3.1.5 Mutation

After the generation of the child mask, it is mutated as per Algorithm 3. A new candidate architecture is selected out of the template network with the activated blocks corresponding to the set bits in the mutated mask, as explained in Section 3.1.1.
We recruited six individuals for this design study; three of them had less than a year’s experience (non-experts) and three had more than two years of experience in deep learning (experts) and three had less than a year’s experience (non-experts). More details about the participants are provided in the Section 6.1.1.

4.2 Method
Each participant was interviewed about their experience in deep learning and designing neural networks. We also conducted a task-based survey where the participants were asked to experiment with a NN architecture search problem. To begin, we explained to them the operation of the one-shot neural network architecture search technique. This was followed by a set of tasks where the participants were asked to perform an architecture search on a VGGNet template for a Fashion MNIST [67] dataset. A template VGGNet was provided to the users to work with and feedback was collected while the participants were performing the task. The feedback from all participants was used to curate a set of tasks that we aimed at solving with our interface.

4.3 Design Requirements
The feedback from the participants was categorized into four main tasks, as follows:

**T1: Template models:** Every participant pointed out that when starting off with training a neural network for a particular dataset, the first step is to try out the existing state of the art networks and visualize how each component of the network actually helped in the designated machine learning task.

**T2: Drag and Drop Interface:** Every participant mentioned in their feedback about how tedious it can be to code and alter different components of a neural network, especially when the networks are very deep. They suggested that an interface that could make it easier to create and alter the neural network components will be useful.

**T3: Exploration and Comparison of Subgraph structures:** All the participants suggested that it’ll be useful to see how the search space of subgraphs is distributed with respect to their evaluation accuracy. Analysts should be able to compare and explore the subgraph structures in this search space.

**T4: Semi-automatic nature of architecture search:** Four out of the six participants, when introduced to the one-shot architecture search technique, mentioned that infusing some domain knowledge while performing the search will be useful. For example, the evolutionary search algorithm provides evidence to remove a dropout layer from a network; in addition to that, if that user wants to add a batch normalization layer to the network, they should be able to do that in midst of the search iterations. They mentioned that such decisions can come from the combination of domain knowledge and search results.

T5: Adding transparency to search: All the participants suggested that given the nature of the one-shot architecture search algorithm, it will be useful to know which set of sub-graphs have been evaluated by the search at a given iteration. The users should be able to control the direction of search, i.e. select the next set of children sub-graphs to be evaluated.

5 The Interface
Our Interface is specifically designed to make the process of designing and searching through very deep neural network architectures more effective. The first section is the lego view (refer to Figure 1) which is designed for tasks T1, T2, and T4. Lego View not only allows the analyst to readily create and edit a neural network architecture but it also provides a quick detailed visualization of each component in the network. The information in the Lego View is summarized by the Graph View (refer to Figure 1). This view presents the complete architecture of the neural network in the form of a graph, facilitating the visualization of very deep networks on a small screen. In addition, there is also the Search Space View displaying a scatterplot layout with each dot representing a subgraph from the Graph View. Each dot is a candidate neural network architecture (subgraph) and hence this view is a projection of the complete search space of candidate neural networks. When the one-shot search algorithm is functioning, some of these dots are colored based on the accuracy score of the corresponding neural networks as they are evaluated, thus satisfying task T3. The analysts can select a region in this scatterplot to make the search choose candidate architectures from a given region (as shown by the grey area in the Search Space View, Figure 1), referring to task T4.

Our interface is implemented on a python flask server [1] using the pandas [39] and pytorch [49] libraries for training the template and candidate neural network architectures on the go. The frontend is created using ReactJS [2] and D3.js [8]. Every time the analyst edits the neural network architecture in the Lego View or the Graph View, the structure of the neural network is sent to the backend server. The neural network is automatically retrained on the linked dataset after every edit. This is followed by the one-shot architecture search algorithm which returns the fitness values for each of the blocks in the template neural network. The results are then displayed on the frontend.

Algorithm 3 Mutation algorithm

1: procedure MUTATION(newModelMask)
2: mask ← Uniformly generated numbers from 0 to 1
3: mutationRate ← 0.1
4: childMask ← newModelMask * (mask > mutationRate) + (1 − childMask) * (mask <= mutationRate)
5: return childModel

3.1.6 Get Mask Probabilities
At each search iteration, when the new population is updated, each block in the template network is given a fitness score between 0 and 1. This is done by counting the number of times each block appeared in the candidate architectures divided by the population size. Hence, if a block appeared in every neural network in the population, it’s fitness score will be set to one.
Fig. 6. Parameters required by a 2D convolutional layer in a neural network. This example shows a Pytorch [49] class for creating a 2D convolutional layer.

5.1 Lego View

Lego View is the interface supporting many features including visualizing the components and their parameters, editing the components, addition, and removal of the neural network components. Because one-shot neural network search works by scoring different blocks inside a single layer, the architecture design of the lego view is carefully chosen to accommodate ease of handling of these neural networks with multiple repeated components inside a single layer. This is achieved by the addition of blocks inside each layer of the neural network as shown in Figure 5. So each layer can hold multiple blocks and similarly each block can hold multiple components, following the basic hierarchy of Layers > Blocks > Components.

Blocks are the parts of the neural network which are evaluated when the analyst performs the one-shot architecture search. For example, as shown in Figure 5, a layer contains two blocks with the first block containing a MaxPool component and the second block containing the Convolutional and ReLU components. This design means that the analyst wants to evaluate these two choices (blocks) on the dataset, thus deciding whether to remove one of the blocks or to keep both of them. Furthermore, every component inside a block is not compared against each other and is treated as a whole series of operations. As shown in Figure 5, the two components of the second block mean that for any input $x$ to this layer, the comparison will be between $\text{maxpool}(x)$ and $\text{relu} \circ \text{conv}(x)$ operations while evaluating. If the analyst wants to evaluate the components separately, they will have to be put inside different blocks.

Hence each layer can contain multiple blocks that will be ranked against each other by the search algorithm. Analysts can change the depth of the network by adding and removing layers as shown in Figure 5. Also, in the case of adding a layer, a block or a component to the network, the analyst is provided with a list of components that can be added, as shown in section F of Figure 1. Choosing a component $X$ while adding a layer will add a new layer with a single block containing the single component $X$. Adding a block with a component $X$ will add a new block with a single component $X$. Adding a component $X$ will add this component inside the same block after the existing components. Every layer, block, and component is draggable and can be placed anywhere in the network, thus making it easy for the analyst to alter the network structure on the go.

One important aspect while creating a neural network is the parameters of each component. For example, in Pytorch [49], a convolutional layer creation requires several parameters as shown in Figure 6. These parameters can be input inside the lego view in the parameter text field in front of each component’s name separated by commas, shown in Figure 5. It is important that the parameters are validated since the output of each layer has to match the input of the next layer in a neural network. This validation is automated in our interface and as the analyst enters the parameters, the input is validated and is corrected to the nearest integer in case the entered value is incorrect. Also, if the analyst doesn’t wish to enter the parameters of some of the components, the interface will automatically try to fill in appropriate values to complete the network as per the designed architecture.

5.2 Graph View

The graph view acts as a supplement view to the lego view displaying an overall structure of the neural network architecture as a graph, as shown in Figure 7. Moreover, the graph view also supports edit connections between the layers, for example, add skip connections [20] in neural network. In many modern deep neural network architectures like the Resnet [20], the connections between the layers are not linear, but some layers might be connected to another layer after the next layer. In such cases, the architecture needs to be designed with the graph view where the user can edit the connection between the nodes and then enter the parameters for each component accordingly in the lego view. The graph view also supports block highlighting for easy maneuvering inside a large neural network architecture.

Overall, the graph view summarizes every component and the block inside the neural network by using different color coding for each of the different components. Also, the subgraphs, i.e. the dots inside the search-space view can be visualized with a mouse hover; it highlights the blocks present in that subgraph in the graph view. Also, a hover over the block label inside the Block Information View will highlight corresponding blocks inside the graph view, for better placement visualization.

5.3 Search Space and Block Information View

The main purpose of the Search Space View is to visualize the complete subgraph (candidate neural network architectures) search space and cluster it based on evaluated accuracy. The projections in the search space view are obtained by using dimension reduction on the graph edit distance matrix [3], as shown in Figure 8. To start off, a subset of candidate architectures are selected in the form of subgraphs of the supergraph from the graph view. These candidate neural network architectures are subgraphs with nodes labeled by the component types, e.g: C for convolution, R for relu, etc. Using these labeled directed subgraphs, the distances between each of the sampled candidate architectures are calculated using the graph edit distance [3] and are stored in a distance matrix. This distance matrix is then passed onto a t-SNE algorithm [38] to reduce the dimensions of these subgraphs to 2. These 2-D projections are then visualized inside the search space view scatterplot.

The search space view supports several user interactions. With a hover over each dot on the scatterplot, the corresponding sub-graph is highlighted on the graph-view (an example is shown in Figure 1) boxes colored grey in the graph view). Also, as some of the subgraphs are evaluated by the one-shot search algorithm, they are then displayed in this view with a color value showing the evaluated score for that subgraph based on its accuracy on a validation dataset. These colors are chosen according to a color scale (shown below the scatter plot).
As per these colored subgraphs signifying their performance, analysts can drag a region in the search space based on their preference to make the one-shot search choose candidate architectures coming from that region. For example, the grey region in the search space view inside Figure 4 will highlight a union of the set of blocks (in the graph view) present in the search space globally.

Besides facilitating search space pruning, the drag interaction also supports visualizing characteristics of the search space. This information is useful to diagnose the subgraphs in a selected region or to compare different regions of the search space as it presents an overview of a region as evaluated by the search algorithm. The search space view allows some set operations (Union, Intersection, and Complement buttons as shown in Figure 9) to present which blocks are more common in different regions of this space. For example, a drag on the scatterplot to select a subspace and pressing the Get Union button on the navbar will highlight a union of the set of blocks (in the graph view) present in candidate neural networks inside the dragged region (Figure 9). These basic set operations help the analysts to visualize different regions in the search space globally.

At the bottom of the scatterplot is the Block Information View as shown in Figure 10. This view presents additional information about the blocks from a region of the search space in the form of line-charts which are updated at each search iteration. One of the plots is the frequency chart for each block in the search space and the other is the evaluated fitness score for each block. Here, frequency measures the number of times a block with a certain ID appears in the subgraphs of the selected search space region. The user can hover over a mark on the line chart to highlight the corresponding block in the graph view. In this particular example a user would probably remove block #2 due to its low fitness score. Since this seems to be a common block (high frequency) removing it will speed up future evaluations greatly.

6 Evaluation

In this section, we evaluate our interface for its effectiveness in finding the best neural network architectures for three different datasets, namely CIFAR10 [28], MNIST [34], and Retinal optical coherence tomography (OCT) [17]. In the first step, to evaluate how effective our interface is to learn and adapt for beginners and experts in the field of machine learning, we conducted a user study explained in Section 6.1. The user study was conducted on the MNIST dataset. This is followed by a couple of case studies testing our interface on two different datasets. The datasets are chosen to represent very different applications of neural networks, i.e. object detection and medical imaging. We tested the classification accuracies of the neural networks obtained from our interface with the existing state of the art convolutional neural network models, i.e. AlexNet [29] and VGG16 [59]. Refer to the following sections for detailed information on each of these evaluation steps.

6.1 User Study

Our interface was developed as per the requirements listed by the experts working in different areas of deep learning. To further evaluate that the interface is up to the mark in interactively finding the best neural network architectures, six different users were asked to find the best convolutional neural network based on their understanding of deep learning on the handwritten digits (MNIST) dataset. We noted the time taken by each user to finalize a network architecture and the accuracy of the chosen network compared to the original LeNet [33]. After the network search was complete and the users were satisfied with the results, we collected feedback from the users about the usability of the interface and the search process.

6.1.1 Participants

There were six different participants in the user study. These six participants were chosen carefully to cover the categories of experts and non-expert users. Three participants were experts, i.e. with 2-4 years of experience in computer vision research. The other three participants were non-experts, i.e. <1-year of experience in machine learning. The experts were Ph.D. students working in the area of computer vision and non-experts were graduate students who had taken a course in machine learning. Since the users of this interface are expected to have a general understanding of how neural networks work, the users with no experience in machine learning were not selected for this study. 4 females and 2 males composed the six users selected for the study with an overall age band of 22-28 years.

6.1.2 Method

All six users were provided an initial template of an overcomplete LeNet as shown in Figure 11. The network was carefully chosen so that the users could leverage both their domain knowledge as well
as the one-shot search algorithm used in the interface. After a short demo explaining the interface, the users were asked to search for the network which they think will best perform for the MNIST dataset. The participants were asked to access the application on their browser and each of the filtering steps and interactions was logged to a file as the users searched for the best neural network architecture. Each computation was done on a server with a Tesla K80 GPU and 12 GB RAM at the backend.

### 6.1.3 Results

Four important factors were documented to evaluate the performance of the users in the assigned task of finding the best neural network architecture for the MNIST dataset. These factors comprised of (1) the expertise of the user, (2) accuracy of the final model chosen by the user, (3) time taken by the user to search for the model architecture and (4) number of filtering steps the user had to take to complete the search. The accuracy of the customized model (shown in Figure 11) on the test dataset after training for three epochs was 95.46%. This accuracy was treated as the baseline for this user study and the users were expected to search for the model with higher accuracy than the baseline after training their model for three epochs. Also, a well studied model called the LeNet [33] achieves a test accuracy of 98.49% on the MNIST dataset. This accuracy is used to evaluate how good the searched network is for each user, i.e. the closer the accuracy of a searched model is to LeNet, the better the searched model. The results obtained from the user study are shown in Table 1.

| User          | Accuracy of the final model (%) | Time Taken (mm:ss) | Number of filtering steps |
|---------------|---------------------------------|--------------------|---------------------------|
| Expert 1      | 99.23%                          | 14:23              | 5                         |
| Expert 2      | 99.59%                          | 10:39              | 8                         |
| Expert 3      | 98.49%                          | 15:10              | 5                         |
| Non-Expert 1  | 98.09%                          | 35:15              | 15                        |
| Non-Expert 2  | 99.17%                          | 29:35              | 11                        |
| Non-Expert 3  | 98.24%                          | 32:23              | 11                        |

As we can see, all of the expert users were able to find a perfect LeNet from the template network which required the insertion of a MaxPool Layer after the ReLU operation at Layer 2 and Layer 4. Also, the users had to select which of the convolutions worked best at each layer. Expert users took on average 13 min and 24 sec to complete the search taking on average 6 filtering steps in the process. All the experts first started with a search to choose the best performing convolutional kernel at each step. This was followed by experimentation to add new blocks in the network. All the experts were able to use their expertise to achieve an equal or greater accuracy than the original LeNet by coming up with deeper models and the use of more advanced features like Batch Normalization. However, in the case of non-experts, the average accuracy of the final models was slightly lower or equal to the LeNet. The average time taken by non-experts was 32 min and 24 sec, which was considerably higher than the experts because they spent more time searching for the components to add to the network given their lack of expertise. However, despite the lower expertise, the accuracy of the final models listed by non-experts was very close to that of the LeNet. This study hence shows that our interface is an effective tool for experts as well as non-experts to find the best or close to the best neural network for a given dataset.

### 6.1.4 Feedback

All the users after evaluation were asked to provide voluntary feedback about their experience using our interface. All three experts appreciated the template models provided in our interface as a good starting point towards neural network architecture search. They were able to converge to a good performing candidate neural network architecture in a short amount of time by pruning the search space after a few iterations of the evolutionary search algorithm. Moreover, they were able to use the block fitness scores provided in our interface to decide which blocks to edit for better results. On the other hand, two of the non-expert users suggested creating a recommendation system that should suggest the components/layers to edit, based on the search results. All of the users commented about the drag and drop neural network design interface of the Lego View to be user-friendly.

### 6.2 Case Studies

We evaluated our interface with case studies derived from two domains where deep neural networks have been proven to be efficient, i.e. Object Classification [72] and Medical Imaging [35]. To evaluate our interface on the task of Object Classification, we used the CIFAR10 dataset because it is well studied in the area of Computer Vision. CIFAR10 has 80 million images of objects labeled into 10 classes. Similarly from the medical imaging domain, we used the Retinal optical coherence tomography (OCT) dataset. OCT dataset is a collection of 84,495 X-Ray images of the retinal cross-section labeled as four categories of NORMAL, CNV, DME and DRUSEN [27]. Two domain experts were consulted to evaluate our interface from these domains. Expert A who evaluated the interface on CIFAR10 has two years of research experience in computer vision working at a private firm. Expert B who evaluated the interface on the OCT dataset is a researcher working in medical imaging at a hospital with one year of research experience.

#### 6.2.1 Architecture Search on CIFAR10 Dataset

This case study evaluates our interface for its efficiency in creating a template neural network followed by the search of the best performing candidate neural network architecture for the CIFAR10 dataset. Expert A (EA) was first given a short demonstration on using our interface followed by an explanation about the task to be performed. All search steps performed by EA were logged along with the time taken for each search step. EA started off by loading a customized AlexNet [29] into the interface, which has multiple options of blocks to choose from and follows the basic architecture model of AlexNet. This customized version of AlexNet contains the same number of layers as the original network but each layer has multiple blocks. For example, as shown in Figure 11 Layer 1 has multiple convolutional blocks, i.e. Conv 3x3, Conv 5x5, and Conv 7x7 and similarly for other layers. EA explained that the reason for choosing the AlexNet template as a template network was his past experience in using AlexNet for image classification tasks. After a careful understanding of the template AlexNet, EA started out by analyzing the results of the first four iterations of the evolutionary search algorithm, which gives a fitness value for each block of the template network. After the search results from the first four iterations, EA decided to further analyze the 7x7 Convolutional blocks from Layer 1 and 2 because the fitness values of these blocks dropped to zero. Focusing on the search space view, EA was able to find a subspace where the most common block was a 7x7 Convolutional block in Layer 1. EA then dragged this region on the search space view which forced the search algorithm to sample candidate architectures from this search space. After two more search iterations, it was confirmed that the presence of this block actually resulted in a below-par performance of the neural networks; EA decided to remove this block from the search space using the lego view and then continued the search further. Another four iterations of the evolutionary search suggested the removal of 3x3 convolution from Layer 1 and 5x5 convolution from Layer 2; these blocks were removed by EA. Additionally, the search results also suggested that 7x7 convolution was the best at Layer 3 on the evaluation dataset but EA wasn’t convinced about this result because of his past experience. Hence, EA selected a region in the search space view where the most common block was the 7x7 Convolutional block at Layer 3 to evaluate more candidate neural networks from this subspace. It was confirmed after a single search iteration that most neural networks from this search space had high accuracy, hence, giving further evidence that 7x7 convolution was actually the best option among the other blocks at layer 3. The search also suggested
that the linear block with 2,304 input and 4,096 output parameters worked the best at layer 6. This yielded the final architecture of the suggested neural network.

**Results:** EA compared the results of the suggested network with a baseline of AlexNet performance on the CIFAR10 dataset after training for 10 epochs. While the baseline AlexNet has an accuracy of 72.70% on the test data, the network derived from our interface had an accuracy of 74.72%. This accuracy was further improved to 76.34% after EB used his expertise and added an additional batch normalization layer after every convolutional layer in the network. Taking 7 steps to find a good performing candidate neural network i.e. Search -> Prune Network -> Search -> Prune Network -> Add Batch Normalization, it took 32 min 54 sec overall for EA to finish this task. EA was satisfied with the final network since it resulted in better accuracy than the baseline AlexNet model. This study confirmed that our tool can help computer vision researchers to effectively search for and identify high-performing convolutional neural network architectures.

### 6.2.2 Architecture Search on the OCT dataset

This case study evaluated the use of our interface from a different domain of medical imaging. Just like EA, Expert B (EB) was first given a demonstration on using our interface for architecture search. EB then began by loading the OCT [27] dataset, followed by a template model of VGGNet [59]. Similar to EA, EB started off with using a VGGNet template because of his past experience in working with VGGNet and its variations. EB decided to move on with the architecture search by first deciding how many layers to use in the VGG Network, followed by filtering of what layer parameters to use. The four different architectures of different depth tried by EB are shown in Figure 12. After training these networks for one epoch, EB compared the accuracies achieved by these networks on the OCT dataset. The accuracies obtained by each of the VGG Net were 21.88% for VGG11, 28.12% for VGG13, 68.75% for VGG16, and 21.88% for VGG19. Since VGG16 performed the best among these four networks with different depths, EB decided to search for the best neural network architecture starting with the VGG16 template network.

After finalizing the depth of the VGGNet, EB then added an additional 5x5 and 7x7 convolutional blocks to the network at each of the convolutional layers to start testing. After this addition, a new search space visualization was created by our interface. Evaluating this new model for 4 search iterations showed a region in the search space that looked more promising than the rest, thus pointing to a region with candidate architectures performing better than the rest of the regions. After careful evaluation of this region, EB figured out that the most common blocks in this region were the 3x3 convolutions in the first and the second layer. This indeed was confirmed by the fitness values of these blocks being displayed as 1 on the graph view. At this point, EB decided to focus further search iterations on this smaller region of the search space. Moreover, EB removed the 5x5 and 7x7 blocks from the first two layers, further pruning the search space. The same process was repeated for 10 further search iterations to yield the final architecture of the convolutional layers for the custom VGGNet. EB then decided to change a single linear layer in the network to two linear layers by adding another layer before the final linear layer in the network. Finally, EB added a ReLU layer after the newly added linear layer as the activation layer, motivated by his past experience that ReLU activation works well with linear layers on most medical imaging datasets.

**Results:** This new architecture when evaluated by the search algorithm, proved that it worked better than the previous networks with a single linear layer. EB then trained this final network for a single epoch which resulted in an improved accuracy of 75.46% over the original VGG16 network. It took 15 filtering steps and approximately 50 mins for EB to complete this neural network architecture search. Given this positive experience EB recommended the use of our interface for training deep neural networks for medical imaging tasks.

### 7 Conclusion

This paper presents a visual analytics framework to assist in deep neural network architecture search. Our interface combines the automated one-shot neural network architecture search approach with a human in the loop design. This design allows analysts to use their domain knowledge and the one-shot search results to quickly converge to the best performing neural network architecture for a given task. Analysts also have the freedom to apply certain soft constraints at their discretion, for example trading off between neural network size and accuracy, etc. Our interface is also less resource-intensive than conventional automatic neural network architecture search algorithms. Analysts can quickly load a template neural network along with their dataset and explore different subset neural network architectures to find the best one. Our evolutionary search algorithm allows for quick sampling of well-performing candidate architectures which can then be further evaluated for their performance. A design study was conducted in collaboration with several researchers working in the deep learning domain with the goal to lay down the tasks to be performed by our interface. We evaluated our framework for its ability to better search for the best performing neural network architecture with the help of a user study. In addition, we also provide evaluation results from two case studies with experts which show the applicability of our interface in different domains of deep learning.

Several important lessons were learned while designing this framework. Our initial discussion with domain experts was decisive in pinning down the main interface design. For example, our collaborators suggested that a graph view could be useful to better present the structure of the neural network. After all tasks were formulated within comprehensive discussions with the experts, it was easier to design the visual interface and its components. Also, we realized that adding strong user interaction facilities was important, as a means to allow users infuse their domain knowledge into the search process to accelerate convergence to the final solution.

However, besides the effectiveness of our present interface, there still remains some scope of improvement, which will be taken on in future work for this project. First we would like to further improve the search by implementing more advanced algorithms, like reinforcement learning and Bayesian Optimization. Also, it will be helpful to see how the change in the dropout values will affect the generation of new candidate architectures from a given population. During the evaluation, experts suggested having an inverse operation in the search space view where the region in the search space will be highlighted based on the most common block selected by the analysts. These features are not yet supported and we will continue work on our interface to incorporate them in the future.

### References

[1] Python flask. [https://flask.palletsprojects.com/en/1.x/](https://flask.palletsprojects.com/en/1.x/) Accessed: 2020-03-19.

[2] React.js [https://reactjs.org/](https://reactjs.org/) Accessed: 2020-03-19.

[3] Z. Abu-Aisheh, R. Rapeaux, J.-Y. Ramel, and P. Martineau. An exact graph edit distance algorithm for solving pattern recognition problems, 2015.

[4] A. Adadi and M. Berrada. Peeking inside the black-box: A survey on explainable artificial intelligence (xai). IEEE Access, 6:52138–52160, 2018.

[5] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang. Cherrypick: Adaptively unearthing the best cloud configurations for big...
data analytics. In 14th [USENIX] Symposium on Networked Systems Design and Implementation (NSDI ’17), pp. 469–482, 2017.

[6] A. Anand, L. Wilkinson, and T. N. Dang. Visual pattern discovery using random projections. In 2012 IEEE Conference on Visual Analytics Science and Technology (VAST), pp. 43–52. IEEE, 2012.

[7] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le. Understanding and simplifying one-shot architecture search. In International Conference on Machine Learning, pp. 550–559, 2018.

[8] M. Bostock, V. Ogievetsky, and J. Heer. D+ data-driven documents. IEEE transactions on visualization and computer graphics, 17(12):2301–2309, 2011.

[9] A. Brock, T. Lim, J. M. Ritchie, and N. Weston. Smash: one-shot model architecture search through hypernetworks. arXiv preprint arXiv:1708.05344, 2017.

[10] Z. Cao, G. Kuennng, K. Mueller, A. Tyagi, and E. Zadok. Graphs are not enough: using interactive visual analytics in storage research. In 11th [USENIX] Workshop on Hot Topics in Storage and File Systems (HotStorage’19), 2019.

[11] D. Cashman, S. R. Humayoun, F. Heimerl, K. Park, S. Das, J. Thompson, B. Saket, A. Mosca, J. Stasko, A. Endert, et al. A user-based visual analytics workflow for exploratory model analysis. In Computer Graphics Forum, vol. 38, pp. 185–199. Wiley Online Library, 2019.

[12] D. Cashman, A. Perer, R. Chang, and H. Strobelt. Ablate, variate, and condition: Visual analytics for discovering neural architectures. IEEE transactions on visualization and computer graphics, 26(1):863–873, 2019.

[13] M. Cavallo and Ç. Demiralp. Clustrophile 2: Guided visual clustering analysis. IEEE transactions on visualization and computer graphics, 25(1):267–276, 2018.

[14] J. Choo, H. Lee, Z. Liu, J. Stasko, and H. Park. An interactive visual testbed system for dimension reduction and clustering of large-scale high-dimensional data. In Visualization and Data Analysis 2013, vol. 8654, p. 865402. International Society for Optics and Photonics, 2013.

[15] S. Das, D. Cashman, R. Chang, and A. Endert. Beamnes: Interactive multimodal steering, selection, and inspection for regression tasks. IEEE computer graphics and applications, 39(5):20–32, 2019.

[16] T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. arXiv preprint arXiv:1808.05377, 2018.

[17] P. Gholami, P. Roy, M. K. Parthasarathy, and V. Vasudevan. OcTid: Optical coherence tomography image database. Computers & Electrical Engineering, 81:106532, 2020.

[18] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Advances in neural information processing systems, pp. 2672–2680, 2014.

[19] A. Gordon, E. Ethan, O. Nachum, B. Chen, H. Wu, T.-J. Yang, A. Zelenyuk, and D. Imre. Clustersculptor: A partition-based framework for building neural architecture search using reinforcement learning. arXiv preprint arXiv:1806.09055, 2018.

[20] S. Liu, B. Wang, J. J. Thiagarajan, P. T. Bremer, and V. Pascucci. Visual exploration of high-dimensional data through subspace analysis and dynamic projections. In Computer Graphics Forum, vol. 34, pp. 271–280. Wiley Online Library, 2015.

[21] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. Journal of machine learning research, 9(Nov):2579–2605, 2008.

[22] W. McKinney. Data structures for statistical computing in python. In S. van der Walt and J. Millman, eds., Proceedings of the 9th Python in Science Conference, pp. 51–56, 2010.

[23] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781, 2013.

[24] G. F. Miller, P. M. Todd, and S. U. Hegde. Designing neural networks using genetic algorithms. In ICGA, vol. 89, pp. 379–384, 1989.

[25] Y. Ming, S. Cao, R. Zhang, Z. Li, Y. Chen, Y. Song, and H. Qu. Understanding hidden memories of recurrent neural networks. In 2017 IEEE Conference on Visual Analytics Science and Technology (VAST), pp. 13–24. IEEE, 2017.

[26] T. Mühlbacher, L. Linhardt, T. Möller, and H. Piringer. Treepod: Sensitivity-aware selection of pareto-optimal decision trees. IEEE transactions on visualization and computer graphics, 24(1):174–183, 2017.

[27] T. Mühlbacher and H. Piringer. A partition-based framework for building and validating regression models. IEEE Transactions on Visualization and Computer Graphics, 19(12):1962–1971, 2013.

[28] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic. Deep learning applications and challenges in big data analytics. Journal of Big Data, 2(1):1, 2015.

[29] E. J. Nam, Y. Han, K. Mueller, A. Zelenyuk, and D. Imre. Clustersculptor: A visual analytics tool for high-dimensional data. In 2007 IEEE Symposium on Visual Analytics Science and Technology, pp. 75–82. IEEE, 2007.

[30] J. E. Nam and K. Mueller. Tripadvisor*’ND: A tourism-inspired high-dimensional space exploration framework with overview and detail. IEEE transactions on visualization and computer graphics, 19(2):291–305, 2012.

[31] C. Olah, A. Satyanarayan, I. Johnson, S. Carter, L. Schubert, K. Ye, and A. Mordvintsev. The building blocks of interpretability. Distill, 3(3):e10, 2018.

[32] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alch’Buc, E. Fox, and R. Garnett, eds., Advances in Neural Information Pro...
[50] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[51] R. Ramprasaath, D. Abhishek, V. Ramakrishna, C. Michael, P. Devi, and B. Dhruv. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *CVPR 2016*, 2016.

[52] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, vol. 33, pp. 4780–4789, 2019.

[53] D. Sacha, M. Kraus, J. Bernard, M. Behrisch, T. Schreck, Y. Asano, and D. A. Keim. Somflow: Guided exploratory cluster analysis with self-organizing maps and analytic provenance. *IEEE transactions on visualization and computer graphics*, 24(1):120–130, 2017.

[54] D. Sacha, M. Kraus, D. A. Keim, and M. Chen. Vis4ml: An ontology for visual analytics assisted machine learning. *IEEE transactions on visualization and computer graphics*, 25(1):385–395, 2018.

[55] J. Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

[56] B. Schneider, D. Jäckle, F. Stoffel, A. Diehl, J. Fuchs, and D. Keim. Integrating data and model space in ensemble learning by visual analytics. *IEEE Transactions on Big Data*, 2018.

[57] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[58] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[59] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[60] M. Sundermeyer, R. Schlüter, and H. Ney. Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*, 2012.

[61] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[62] L. Theis and M. Bethge. Generative image modeling using spatial lstms. In *Advances in Neural Information Processing Systems*, pp. 1927–1935, 2015.

[63] A. Tyagi, Z. Cao, T. Estro, E. Zadok, and K. Mueller. Ice: An interactive configuration explorer for high dimensional categorical parameter spaces. In *IEEE Conference on VAST*, pp. 23–34, 2020.

[64] A. K. Tyagi, A. Kumar, A. Gandhi, and K. Mueller. Road accidents in the uk (analysis and visualization). In *IEEE Conference on VAST*, 2018.

[65] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018, 2018.

[66] C. Wong, N. Houlsby, Y. Lu, and A. Gesmundo. Transfer learning with neural automl. In *Advances in Neural Information Processing Systems*, pp. 8356–8365, 2018.

[67] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[68] S. Xie, H. Zheng, C. Liu, and L. Lin. Snas: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018.

[69] T. Young, D. Hazarika, S. Poria, and E. Cambria. Recent trends in deep learning based natural language processing. *IEEE Computational intelligence magazine*, 13(3):55–75, 2018.

[70] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pp. 818–833. Springer, 2014.

[71] J. Zhang, Y. Wang, P. Molino, L. Li, and D. S. Ebert. Manifold: A model-agnostic framework for interpretation and diagnosis of machine learning models. *IEEE transactions on visualization and computer graphics*, 25(1):364–373, 2018.

[72] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, 30(11):3212–3232, 2019.

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