MAST: A TOOL FOR VISUALIZING CNN MODEL ARCHITECTURE SEARCHES

Dylan Cashman [*]
Tufts University
Medford, MA 02155
dylan.cashman@tufts.edu

Adam Perer
Carnegie Mellon University
Pittsburgh, PA 15213
adam.perer@cmu.edu

Hendrik Strobelt
MIT-IBM Watson AI Lab
Cambridge, MA 02139
hendrik@strobelt.com

ABSTRACT

Any automated search over a model space uses a large amount of resources to ultimately discover a small set of performant models. It also produces large amounts of data, including the training curves and model information for thousands of models. This expensive process may be wasteful if the automated search fails to find better models over time, or if promising models are prematurely disposed of during the search. In this work, we describe a visual analytics tool used to explore the rich data that is generated during a search for feed forward convolutional neural network model architectures. A visual overview of the training process lets the user verify assumptions about the architecture search, such as an expected improvement in sampled model performance over time. Users can select subsets of architectures from the model overview and compare their architectures visually to identify patterns in layer subsequences that may be causing poor performance. Lastly, by viewing loss and training curves, and by comparing the number of parameters of subselected architectures, users can interactively select a model with more control than provided by an automated metalearning algorithm. We present screenshots of our tool on three different metalearning algorithms on CIFAR-10, and outline future directions for applying visual analytics to architecture search.

1 INTRODUCTION

In less than a decade, artificial neural networks (ANNs) have become the preeminent topic in artificial intelligence and data processing, due to their superior performance in machine intelligence competitions. The increased interest in ANNs in the AI community has coincided with the big data movement, in which the capacity to gather, analyze, and process data has pervaded wide sectors of government, industry, and academia. The end result is that neural networks are used by a much wider audience than ever before. Yet even for data scientists with experience in using machine learning models, choosing an architecture and its corresponding hyperparameters can be a difficult, even mystifying task. While ANNs alleviate the need to find an efficient data encoding, they generally require a handcrafted setting of the number and types of layers, the learning rate procedure, and the parameter optimizer, among other hyperparameters.

Recent developments in metalearning have produced model architecture search (MAS) algorithms that are able to select architectures by various methods including Bayesian Optimization (Snoek et al., 2012) and Q-Learning (Baker et al., 2017). This optimization process is very resource intensive and may not terminate for an unreasonably long time; it is seen as a surprising accomplishment if a competitive, untrained architecture is found within 24 hours (Wistuba, 2017). The outcome of this metalearning is typically a set of well-performing models. Many things can go wrong, however.

[*]Work done while an intern at the MIT-IBM Watson AI Lab

If an architecture search is not given enough resources, it may only train each model one or two epochs, and not be provided enough time to make accurate predictions about what will be the most promising candidate models. It may also be the case that the archictecture search itself is poorly parameterized - a reinforcement learning search with poor choices for balancing exploration verse exploitation might never improve its sampling strategy. If the MAS results in a poor model, it may be hard to know whether it was a result of the data being a poor fit for this type of model, or whether it was due to an issue with the parameterization of the MAS.

In this work, we posit that attribution can be facilitated visual inspection of the data generated by the MAS. In order to produce the top k architectures, a MAS might produce thousands of candidate architectures, apportioning tens of thousands of epochs of training among them. The full set of inferences, accuracies, losses, and metadata associated with each model could provide insight into how the MAS algorithm evolves its model sampling over time. It also contains information on which layer subsequences correlate with endemically poor performance on the dataset, providing hints on how to improve the architectures discovered by the automated search. And lastly, it can enable the analyst to choose a model they find promising that may not have been explicitly recommended by the search, i.e. if they wanted a model with the highest accuracy under a certain limitation of parameter size for deployment on an edge device.

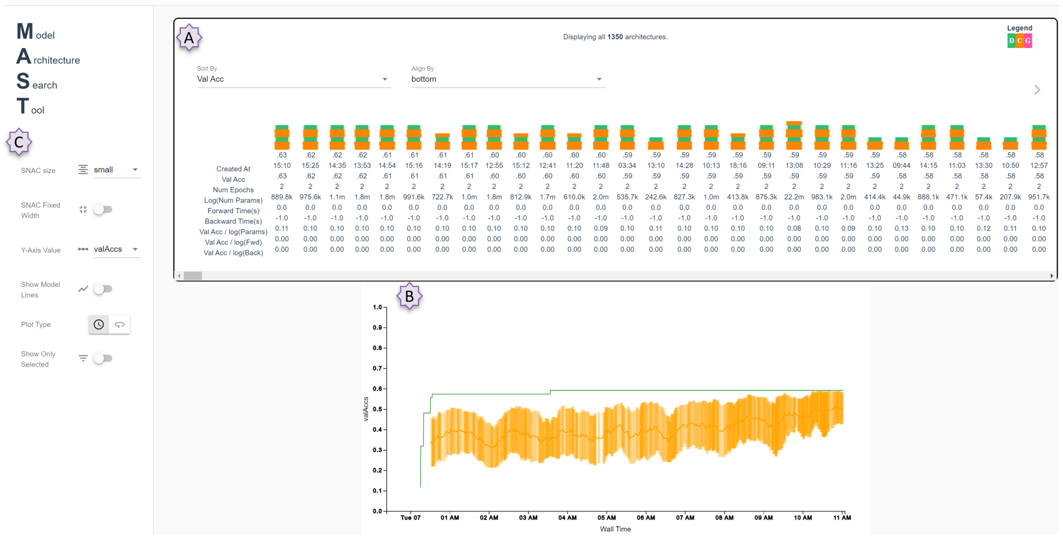
In this work, we present a tool for visualizing data generated during MAS, designed for data scientists that don't have the ability to manually craft their own architecture and so use an automated search. In addition to rendering the process more transparent, our tool helps the analyst verify assumptions about the architecture search using a visual overview. By selecting subsets of discovered models, they can gain an understanding of the effects of different layer subsequences that might then be used to subsequently improve models returned by the MAS. By viewing loss and training curves and comparing the size of models in number of parameters, they are also able to have finer-grained selection from the set of models discovered by the MAS. We demonstrate our tool with screenshots from three MAS algorithms on CIFAR-10 data: a Markov chain with fixed training epochs, the same chain with a bandit-based approach to training, and a reinforcement learning based algorithm. We describe how basic properties of these metalearning algorithms can be verified from the visualizations in our tool, and describe the steps a user might take to better understand layer subsequences and interactively select models. We conclude by discussing some future functionality and planned user evaluations.

2 Model Architecture Search Data

In a typical model architecture search, a model space $\mathcal{M}$ is periodically sampled by a strategy π. For feed forward convolutional neural networks, the focus of this work, the model space is typically the set of all possible sequences of layers (e.g. convolutional layers, fully connected layers, etc.) along with their hyperparameters. The sampling strategy π is typically stochastic: a baseline strategy might uniformly sample from the model space. For each sampled model m_i, an MAS will produce a set of results on training epochs, $\{e_{ij}\}$, where each training epoch result might represent a single scalar value, such as the training accuracy or loss on that epoch, or it might include all inferences on all data in the epoch. When the MAS terminates, it outputs its best guess (or top k best guesses) at the optimal discovered model, m^*. MAS algorithms must trade off between sampling many models and training them for a short number of epochs verse sampling few models but training them sufficiently. Thus, the ranking of models that produces m^* is only an estimate. In addition, the sampling strategy π largely determines which parts of the model space are explored, and a poor strategy might result in many poor models being trained and thus wasting resources. We posit that by viewing the training data of discovered architectures, a user can gain a better understanding of how the MAS might be reparameterized or overruled.

We enumerate three goals that may be served by visualizing the model space in a MAS over feed forward CNNs.

- **G1: Verifying MAS Algorithms**. It is important to analysts to verify and validate assumptions about the MAS algorithm in order to trust that it discovered a large enough variety of models to have produced a good model. The analyst might want to confirm that the sampled models are improving over time, or that the MAS is apportioning enough training to get a realistic estimate of the performance of its sampled models.



SNAC size: small
SNAC Fixed Width
Y-Axis Value: valAccs
Show Model Lines
Plot Type
Show Only Selected

Displaying all **1350** architectures.

Sort By: Val Acc
Align By: bottom

	.63	.62	.62	.62	.61	.61	.61	.61	.60	.60	.60	.60	.60	.59
Created At	15:10	15:25	14:35	13:53	14:54	15:16	14:19	15:17	12:55	15:12	12:41	11:20	11:48	03:34
Val Acc	.63	.62	.62	.62	.61	.61	.61	.61	.60	.60	.60	.60	.60	.59
Num Epochs	2	2	2	2	2	2	2	2	2	2	2	2	2	2
Log(Num Params)	889.8k	975.6k	1.1m	1.8m	1.8m	991.6k	722.7k	1.0m	1.8m	812.9k	1.7m	610.0k	2.0m	535.7k
Forward Time(s)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Backward Time(s)	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
Val Acc / log(Params)	0.11	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.09	0.10
Val Acc / log(Fwd)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Val Acc / log(Back)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

	.59	.59	.59	.59	.59	.59	.59	.59	.59	.58	.58	.58	.58	.58	.58
Created At	13:10	14:28	10:13	18:16	09:11	13:08	10:29	11:16	13:25	09:44	14:15	11:03	13:30	10:50	12:57
Val Acc	.59	.59	.59	.59	.59	.59	.59	.59	.59	.58	.58	.58	.58	.58	.58
Num Epochs	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
Log(Num Params)	242.6k	827.3k	1.0m	413.8k	875.3k	22.2m	983.1k	2.0m	414.4k	44.9k	888.1k	471.1k	57.4k	207.9k	951.7k
Forward Time(s)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Backward Time(s)	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
Val Acc / log(Params)	0.11	0.10	0.10	0.10	0.10	0.08	0.10	0.09	0.10	0.13	0.10	0.10	0.12	0.11	0.10
Val Acc / log(Fwd)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Val Acc / log(Back)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Figure 1: A screenshot of MAST using data gathered during a MetaQNN run over CIFAR-10 data. MAST features a model drawer showing architectures along with various training data (**A**), a zoomable MAS overview (**B**), and a set of controls that modify the visualizations (**C**). MAST visualizes data generated during a Model Architecture Search in order to imbue trust in the model architecture search, and enables an analyst to use their domain knowledge in selecting an architecture.

- **G2: Attributing Layer Patterns**. By selecting and comparing groups of models, the user can attribute layer patterns to performance or model size. This information can then be used to determine if the sampling strategy is sampling the right types of models. Inspecting two different layer sequences with small differences, such as two similar architectures with and without a dropout layer, can lead to a better understanding of individual layer patterns, and can help the user fine tune a model.
- **G3: Improving Model Selection**. There are several reasons why an MAS algorithm might actually produce a m^* that is a poor fit for the analyst's task. First, if the MAS is not provided enough resources, it may not have trained its models a sufficient number of epochs to have much confidence that its m^* is actually a superior model. Second, the analyst may have additional knowledge that they were unable to encode into the objective function of the MAS. For example, they may have a restriction where they cannot accept models over a certain memory size because they must be deployed on edge devices.

While any one of these goals could be formally defined and explored computationally on a dataset, the goal of this work is to make a tool to provide a starting point to engage with these questions. Visualization excels at abstracting away complexity, providing canvas for high-level insights while still allowing for drilled-down details on demandVan Wijk (2005). Humans excel at finding visual patterns, suggesting that **G2** could be addressed via a juxtaposition of visual encodings of model architectures. Humans also excel at seeing temporal trends (**G1**), and within those trends, outliers (**G3**). There are several examples of visual analytics systems empowering users to discover faults in neural models that suggest this is a promising avenue of research (Strobelt et al., 2019; Simonyan et al., 2013).

3 MAST: A MODEL ARCHITECTURE SEARCH TOOL

MAST, seen in Figure 1, is a tool for exploring the data generated during a model architecture search. It has three components. Figure 1A shows the model drawer, a table in which each column represents a single model discovered during the search. The columns hold various information about the model's training, including the number of epochs trained, the best validation accuracy attained, and the number of parameters of the model. Models can be sorted by any attribute, and even certain

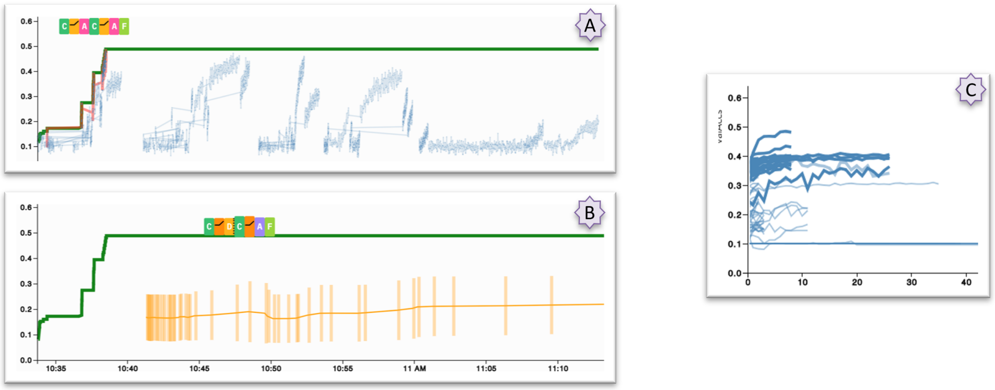


Figure 2: The MAS overview on data generated by the Hyperband MAS, which starts by training many models for a short number of epochs, then recursively continues to train the best subset at each iteration. The configuration options, seen in Figure 1C, let the user select which overview is given. Subfigure A shows the loss curves, demonstrating that some trained models don't improve at all, and also illustrating how Hyperband intersperses training of different models. Subfigure B removes the individual training curves, and shows orange lines representing rolling averages of the variance (vertical line) and performance (horizontal line) of each newly sampled model. The horizontal axis is wall time and the vertical axis is validation accuracy. Subfigure C shows the third possible view, where the performance curves of all models are aligned based on number of epochs rather than wall time.

compositions of attributes, such as the validation accuracy divided by the log of the number of parameters. Such composite attributes can help the analyst see models according to metrics not considered by the MAS.

Each model is represented by a space-efficient visual encoding called a Sequential Neural Architecture Chip (SNAC). SNACs are used to provide a visual shorthand of the structure of the model. They encode the sequence of layers, and can thus enable an analyst to see patterns in subsequences or when certain subsequences occur (beginning, middle, or end). They also encode the dimensionality of the activations flowing through the model - width corresponds to a log scale of the total tensor size at that layer. In this way, analysts can get a sense of whether the model is projecting to a higher dimension, or simply reducing the dimensionality. More information about the design decisions involved in developing SNACs can be found in the appendix.

Underneath the model drawer is the MAS overview. Figure 1B shows one of the possible visualizations for the overview. It shows two indications of performance of the MAS run. The vertical axis measures validation accuracy, and the horizontal axis measures wall time. The green line shows the best validation accuracy attained *to that point*, so that it is monotonically increasing. The green line represents what might be outputted by a MAS were it to cut off at a given wall time. The horizontal and vertical orange lines correspond to rolling mean and variance of the k most recently discovered models. This can illustrate what the MAS algorithm is doing. In Figure 1, the MAS is MetaQNN, a reinforcement learning algorithm typified by an initial exploration stage (with poor average performance and large variance), followed by an exploitation stage (with better performance and decreased variance)(Baker et al., 2017).

In contrast, figure 2 shows the Hyperband MAS, which apportions its training resources much less equally, samples models less regularly, and thus has its orange lines spaced out more. Figure 2 also shows individual model training curves, one of the configuration options in MAST, controlled in the side panel in Figure 1C. By viewing individual training curves, the analyst can determine if a given model was trained sufficiently, or should have been given more training epochs. Analysts can also view performance per epoch in order to compare training curve shapes directly. All views can be used to select subsets of models to compare in more detail in the model drawer.

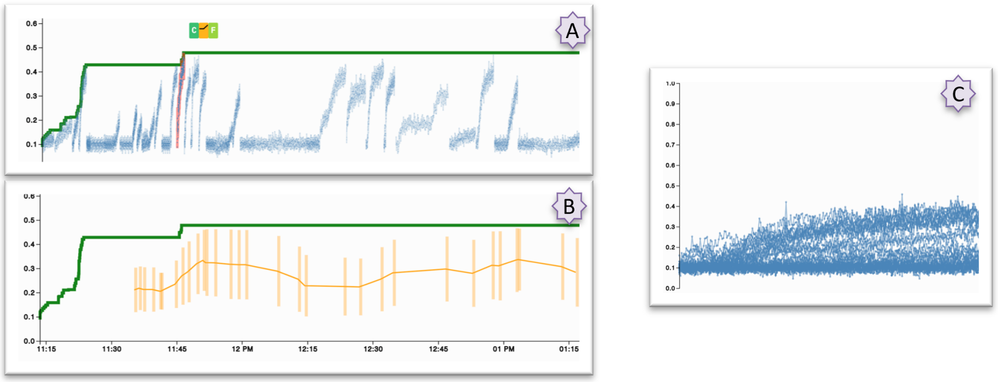


Figure 3: The MAS overview on data generated by a Markov chain with each model trained for a fixed number of epochs. Subfigures A, B, and C are three different overviews available to the user; see Figure 2 for a description.

3.1 Workflow

A user of MAST would typically start by viewing the MAS overview, Figure 1B, to validate assumptions about the MAS (**G1**). For example, a user might look for the exploration / exploitation pattern, expected in Q-learning strategies, when running the MetaQNN algorithm. If the Hyperband metalearning algorithm was used (Figure 2), the user may want to confirm that it was parameterized correctly so that it allocated enough training epochs to determine which were the most promising architectures. Viewing the training curves by epoch, in Figure 2C, the user may note that many of the models were still improving when their training was cut off, and the MAS should be rerun with more resources, or that they should take a large number of the candidate models and train them for longer before choosing an architecture.

The user may note that the MAS performed as expected, but want to check if their sampling strategy π resulted in reasonable models being trained. In figures 3A and 3C, it is evident that the MAS spent a lot of time training poor models. The user can select a set of good models and bad models from those views, and then compare their architectures in the model drawer (**G2**). They may see that certain layer transitions result in poor performing models, and they can then reparameterize their sampling strategy π and rerun their MAS.

Lastly, the user may use MAST to assist in selecting a model. They can view individual loss curves to determine if one model might appear to benefit more from more training. Then, they can select the set of models they are most interested in, and view them in the model drawer, comparing their parameters and time for inference (forward time) and time for training (background time).

4 Discussion

There are many directions for future research. The visual encodings used in MAST are restricted to feed forward CNNs, but more complex neural networks are increasingly being used for domain problems. New visual encodings need to be found that can still be easily compared like SNACs. There could be more support for experimentation as well.

Metalearning is an enticing potential solution to the very real problem of enabling nonexpert analysts to use complex learning algorithms like CNNs. However, it can require an obscene amount of resources, and it may be irresponsible to solve every problem with a metalearning solution. Each metalearning run uncovers a rich set of data. Efficient metalearning algorithms make use of this data possibly by updating their sampling strategies or adapting discovered models. But that data can be useful also in instructing the nonexpert in the task of manually constructing a model, so that metalearning might not be needed the next time. It may also be helpful in providing a sanity check that the automated process produced the desired output. Whenever rich data is generated, it can only help to provide visualizations to help interpret that data.

Acknowledgments

This work was partially supported by DARPA grant FA8750-17-2-0107 and the IBM Research Cambridge internship program.

References

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *International Conference on Learning Representations*, 2017.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pp. 2951–2959, 2012.

Hendrik Strobelt, Sebastian Gehrmann, Michael Behrisch, Adam Perer, Hanspeter Pfister, and Alexander M Rush. Seq2seq-vis: A visual debugging tool for sequence-to-sequence models. *IEEE transactions on visualization and computer graphics*, 25(1):353–363, 2019.

Jarke J Van Wijk. The value of visualization. In *VIS 05. IEEE Visualization, 2005.*, pp. 79–86. IEEE, 2005.

Martin Wistuba. Finding competitive network architectures within a day using UCT. *arXiv preprint arXiv:1712.07420*, 2017.

Supplemental Material
SNACs: A Visual Encoding for Sequential Neural Network Architectures

Dylan Cashman
Tufts University
Medford, United States
dylan.cashman@tufts.edu

Adam Perer
Carnegie Mellon University
Pittsburgh, United States
adam.perer@cmu.edu

Hendrik Strobelt
MIT IBM Watson AI Lab
Cambridge, United States
hendrik.strobelt@ibm.com

ABSTRACT

Convolutional Neural Networks (CNNs) and other sequential artificial neural networks are used extensively in many domains because of their ability to fit many different types of problems over unencoded data, such as images or audio. While such models are deployed extensively, no visual encoding exists that can facillitate comparison and inspection of multiple architectures in a space-efficient way. In this work, we introduce a new visual encoding, Sequential Neural Architecture Chips (SNACs), that conveys layer types and order, upsampling and downsampling, as well as skip connections. Using the plate notation of graphical models, SNACs can encode very deep networks such as ResNet using a minimal amount of screenspace. SNACs are expected to be used in both static formats such as publications, blog posts, and technical documentation, as well as dynamic formats such as visual analytics systems. SNACs will be released in an open source code repository as well as a web tool for constructing SNACs.

INTRODUCTION

As Artifical Neural Networks (ANNs) become more prevalent, they become communicated in many different forums and media. When a neural network is presented in a formal writeup, it is typically described with both a textual description of the hyperparameters used as well as an illustration of the topology of the network. This standard description is a poor fit, however, for many other communication scenarios, such as a high-level report, or usage in a visual analytics system. Instead, there exists a need for a visual encoding of neural networks that efficiently trades off space for information.

In this work, we present such a visual encoding, called Sequential Neural Architecture Chips (SNACs). SNACs encode the sequence of layers found in a sequential neural network, as well one other property about each layer, such as the size of the data flowing through the layers, or the number of parameters. By using plate notation popular in statistical graphical models, they are able to portray very deep networks that have repetitive blocks, such as the popular Residual Network used for computer vision tasks. In an interactive setting, they also provide details-on-demand via mouse hover, and can be toggled between various sizes and shapes. They can be used in academic publications, blog posts, and dynamic visual analytics systems.

VISUAL ENCODINGS FOR NEURAL NETWORKS

ANN architectures are traditionally shown one at a time, either as part of a figure in a publication or in a detailed graph view in an application. While these encodings are effective, they have severe limitations in their usage. A figure typically takes up a large portion of a printed page, and in an application such as Facebook's Activis [6] or Google's TensorBoard [1] a single network is the focus of the entire application. These encodings take up so much space because they aim to communicate to an expert user as much available information about the corresponding network. Table 1 shows a non-exhaustive set of previous visual encodings of neural networks. A trend is apparent where earlier encodings visualize the full detail of the network's topology, such as the individual cell connections, while later encodings abstract such information away.

The designer of a visual encoding must make two types of decisions: which data features to visualize, and what visual channels to which those features map [2]. Encodings that visualize too many features can suffer from complexity or space demands that limit the practicality of the encoding, as in the examples above, and can require significant cognitive load from the viewer. Thus, the designer must make a tradeoff between the coverage of data supported by an encoding, and the versatility of the resulting encoding.

Before creating an entirely new encoding, it is necessary to review currently existing encodings to see if a solution already exists. Popular encodings for network architectures are summarized in Table **??**. The cell-level computation graph increases its usage of space linearly with the number of nodes in the network, so it would be impossible to visualize multiple models, if not even a single deep convolutional model, and would not allow for multiple resolution views. Similarly, the layerwise-verbose illustration and layer-wise filter illustration face scaling issues. Although they reveal very interesting details, such as the filters of the model, that might be useful in comparing one model to another, our system hopes to allow the user to compare dozens or even hundreds of models at once. The Tensor-level computation graph is able to describe almost any possible neural network, and might be able to address scaling by vigorously applying aggregation methods, but its layout does not facillitate easy comparison.

The previous encodings outlined in Table 1 mostly convey a large amount of information, and thus are limited in their versatility. In contrast, we aim to find an encoding that is highly versatile and space-efficient. We also want an encoding that can be used as a component of a larger visual analytics system. In particular, we want an encoding that supports the following constraints.

- **C1: Visualizing multiple models concurrently for comparison.** A user may want to compare models for the sake of model selection or comparison. An effective visual encoding will not only need to be small enough to fit multiple architectures on the same page or screen, but it will need to support paradigms for visual comparison such as juxtaposition, as suggested by Gleicher [4].
- **C2: Compatible with multiple levels of detail depending on screen space and user guidance.** The user might want to view an overview of dozens of models to view large-scale patterns between models. It is also likely that a system would allow the user to drill down to compare some subset of the models, or even to inspect a single model in depth. Different visual encodings could be used, but it would be better if a single visual encoding could comfortably transition between different resolutions, in order to maintain the user's mental model.
- **C3: Encode as much relevant information as possible.** Any visual encoding designer must decide on the subset of features that they provide a visual mapping for. For deep models, there is an enormous amount of potential features to visualize, such as the individual layer activations and weights, the layer sizes, the layer order, skip connections, and various hyperparameters like the learning rate. The more attributes that are visualized, the greater the visual complexity of the of the resulting encoding. The encoding must trade off between coverage of data and portability of usage of the encoding, based on the intended usage.
- **C4: As large coverage of model architectures as possible.** An ideal encoding should completely cover the space of network architectures. However, as with **C3**, covering more architectures conflicts with other design goals. Thus, our encoding must cover as many architectures as possible without sacrificing our other constraints.

SEQUENTIAL NEURAL ARCHITECTURE CHIPS (SNACS)

The simplified computation graph shown in Table 1 and used to describe He et. al's popular ResNet architecture [5] amongst others, satisfies many of our needs. In that encoding, similar to the tensor-level computation graph, each node represents a transormation of the data flow. However, the computation graph is simplified by removing a lot of information of the architecture, including the data size and some hyperparameters. It features a compact design with a fixed width and a simple layout. However, we make some modifications to be more efficient with screen space and to enable comparison.

First, we choose to only support sequential models with skip connections. While cutting-edge networks found in the research literature include many non-linear topologies such as U-networks, a significant portion of models in production are based off of popular image detection CNNs such as ImageNet[8] or ResNet[5]. Sequential networks can be naturally aligned along a single dimension, enabling easy comparison along the orthogonal axis. This makes the edges between nodes mostly superfluous, since adjacency of layers can imply connections between them. In contrast, supporting non-sequential models necessitates supporting any potential network topology, and there is no visual encoding that allows for a unique linear layout of arbitrary graphs.

Second, by writing the names of layers, such as "Conv", "MaxPool", and "LocalRespNorm" while double-encoding the layer types via color, the simplified computation graph wastes space, and makes graph's nodes larger than needed. Finally, there are visual channels in this encoding that are still left unused; the size of each node could be used to encode some parameters.

With these considerations in mind, we present our visual encoding for network architectures, SNACs. SNACs are able to visualize most sequential neural network models, and can display them at several different sizes.

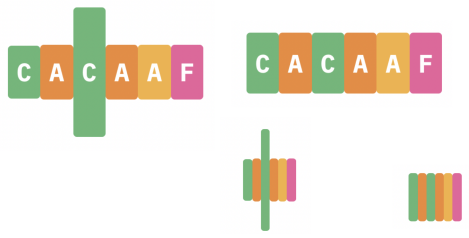
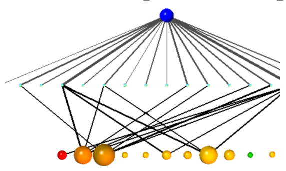
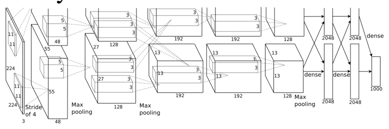
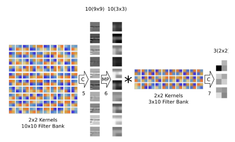
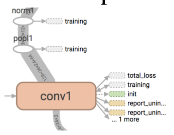
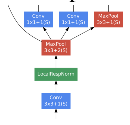


Figure 1. The *SNAC* visual encoding for neural network architectures. This model has a two convolutions, each followed by an activation, and concludes with an average pooling layer and a fully connected layer. Each block represents a layer in a sequential CNN. Color encodes the type of layer, while width can be used to encode a numeric attribute such as tensor width. Additional attributes can be made available via tooltip if SNAC is deployed in a web application. SNAC functions at several different sizes and fixed widths.

The primary visual encoding in a SNAC is the sequence of types of layers. This is based on the assumption that the order of layers is the most important distinction in architectures, considering that all previous encodings, including those outlined in Table 1, all displayed this information. Layer type is redundantly encoded with both color and symbol. The symbol is the first letter of the name of the layer. Beyond the symbol, some layers have extra decoration. Activation layers have glyphs for three possible activation functions: hyperbolic tangent (*tanh*), rectified linear unit (ReLU), and the sigmoid function. Dropout also features a dotted border to signify that some activations are being dropped. The second type of data encoded by an archip is the size of the data flowing through the network, since this is the second most frequently encoded piece of information in existing visual encodings of network architectures. The height of each block corresponds to the data

Table 1. Visual Encodings for neural networks.

	Source	Description	Encodings
Cell-level Computation Graph 	[11]	Each cell in inference is represented as a node, activations can be encoded via size or color.	activation size, layer types, filter and pool size, architecture sequence
Layer-wise Verbose Illustration 	[13, 8, 9]	Each layer is shown side-by-side, and the distances in the diagram roughly map to the size of layers and filters.	activation size, layer types, filter and pool size, stride, architecture sequence.
Layer-wise Filter Illustration 	[7]	Each layer is shown side-by-side, and the layers are encoded as the learned filters.	activation size, layer types, filter and pool size, filter and pool values, architecture sequence.
Tensor-level Computation Graph 	[6, 12]	Operations in inference are represented as nodes, and data flow typically is encoded in edges.	activation size, layer types, filter and pool size, architecture sequence, data flow size.
Simplified Computation Graph 	[10, 5]	Each layer is shown in a directed graph as a fixed width glyph.	layer types, filter and pool size, stride, architecture sequence.

size on a bounded log scale, to indicate to the user whether the layer is increasing or decreasing the dimensionality.

Many solutions are optimally solved using very deep networks, such as VGGnet or Resnet, and yet such deep networks require rendering repeated modules and redundantly using screen space. To address these types of models, archips make use of plate notation, a technique from the machine learning literature for expressing repeated dependencies in graphical models [3]. Whenever a pattern is repeated N times contiguously in a sequence of layers, that pattern is extracted as a *plate*, and the number of repetitions is displayed to the user. Archips use regular expressions to find identical patterns in sequential architectures so that very deep models can be displayed as succinctly as possible. An example can be seen in figure 3. Some optimizers are also able to create networks with skip connections, in which output of one layer is fed to both the succeeding layer and a layer in the future. Archips can represent skip connections via internal, semi-transparent arrows, as seen in figure 2. While there is a scaling issue with internal arrows, allowing for external arrows would cause the visual footprint of an archip to grow outside of its rectangular bounding box, violating the constraints **C1** and **C2**.

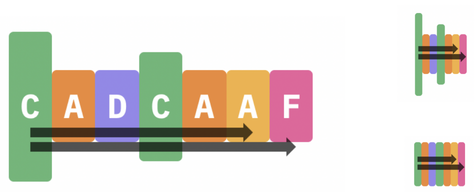


Figure 2. SNAC has support for skip connections, which are prevalent in many sequential CNNs, such as ResNet.

In order to satisfy the need for multiple levels of detail depending on screen space and user guidance (**C2**), archips have several different layouts, including fixed with, variable width, and squished forms. These multiple forms can fit many different macro layouts within an application in order to facillitate comparison between many models (**C1**). They are compact and succinct while being able to represent all sequential CNN models (**C4**) and display parameters about each layer, such as tensor size or dropout rate (**C3**).

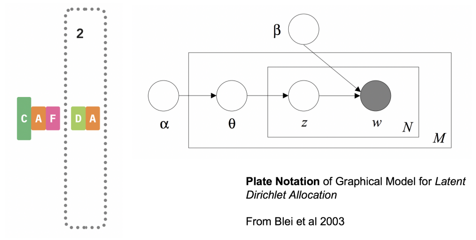
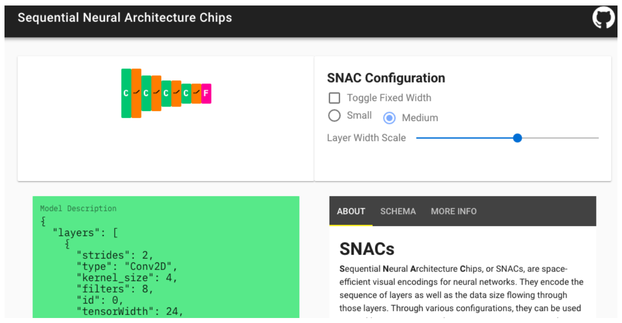

2
C A F D A
β
α
θ
z
w
N
M
Plate Notation of Graphical Model for *Latent Dirichlet Allocation*
From Blei et al 2003

Figure 3. Many large CNNs can be visually encoded in a small space by taking advantage of repetitions in their sequential layers. In this figure, the last two layers we see are actually repeated twice. To visualize this repetition, SNAC makes use of a method from graphical models in the machine learning literature called *plate notation*.

Online tool



```
Model Description
{
  "layers": [
    {
      "strides": 2,
      "type": "Conv2D",
      "kernel_size": 4,
      "filters": 8,
      "id": 0,
      "tensorWidth": 24,
```

SNACs

Sequential Neural Architecture Chips, or SNACs, are space-efficient visual encodings for neural networks. They encode the sequence of layers as well as the data size flowing through those layers. Through various configurations, they can be used

Figure 4. SNACs can be generated via a webpage by uploading a JSON description of a sequential model. https://www.eecs.tufts.edu/~dcashm01/snacs/

The code base for SNACs will be made open source. However, in order to facillitate easy usage of the visual encodings, especially for casual usage in blog posts or in publications, it is important that they be trivially easy to produce. For this reason, we have developed a single page web application, as seen in Figure 4 that allows users to upload descriptions of sequential models in JSON that automatically generates SNACs.

EXAMPLE USAGE

SNACs can be used as a colorful component of figures in deep learning literature, as made available in the tools described in the previous section. However, its use in visual analytics application was a significant factor in the design decisions made. In this section we provide some examples of how a SNAC could be used in a visual analytics application.

Figure 5(a)-(c) are all views within a different tool being developed to understand model architecture searches for automated machine learning. In Figure 5(a), SNACs are used in a *model drawer*, a sorted, filtered selection of model architectures. SNACs here have a fixed width shadow, but variable widths, to allow a user to quickly view whether a model is embedding data in a higher or lower dimensional space. SNACs take advantage of the space allotted to let users visually compare and contrast both the sequence of layers and the size of the data passing through the layers. Figure 5(b) shows the evolution of validation accuracy of multiple randomly spawned models by wall time. When the user mouses over a particular model's validation accuracy curve, its corresponding SNAC pops up in a tooltip to show the user some information about that model's architecture in a concise, unintrusive way. Lastly, Figure 5(c) demonstrates how SNACs can be used as glyphs. It portrays several hundred models in a 2-D scatter plot depicting validation accuracy vs. wall time. The model's SNAC, and some text giving its validation accuracy, act as a glyph to provide the user of some sense of the models being embedded in the model architecture search process. While this tactic suffers from severe overplotting, it illustrates the strength of a visual encoding for neural network architectures that are space-efficient.

CONCLUSION

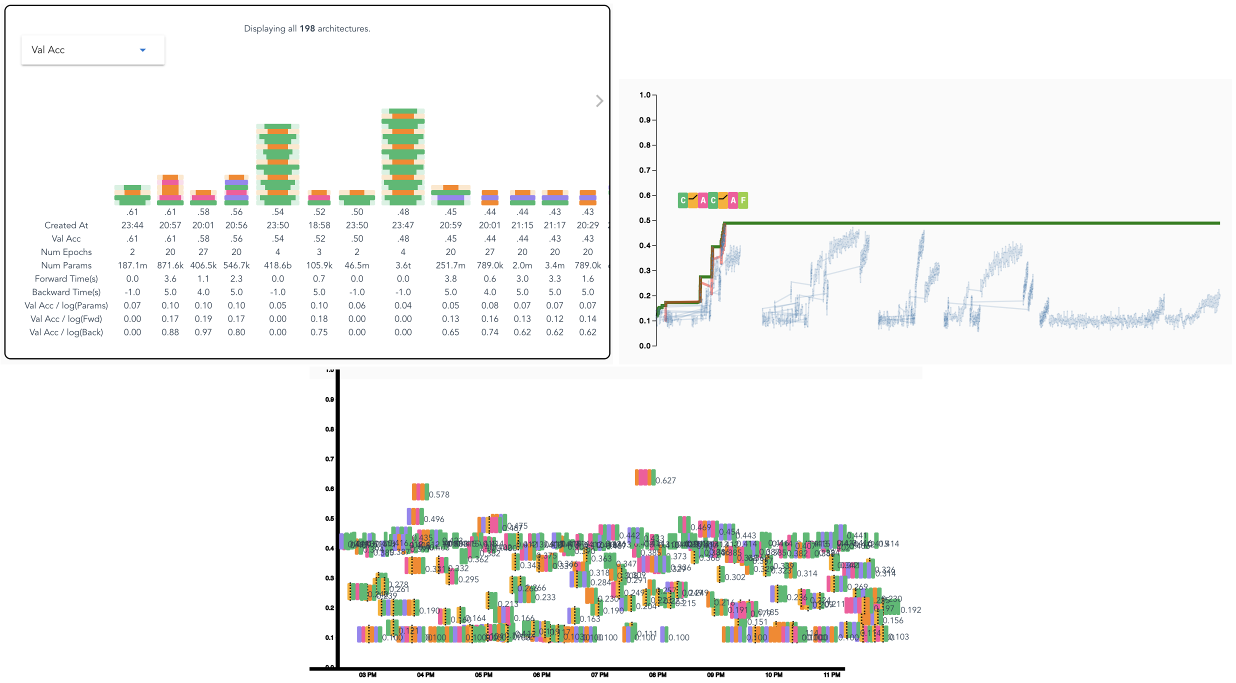
In this work, we describe SNACs, a space-efficient visual encoding for sequential neural network architectures. SNACs trade off less information for much more versatility than existing visual encodings for neural network architectures. They are released in several public tools for use in publications, blog posts, and interactive tools. We also provide several examples of SNACs in use in visual analytics applications in current development.

ACKNOWLEDGMENTS

This work was partially supported by DARPA grant FA8750-17-2-0107 and the IBM Research Cambridge internship program.

REFERENCES

1. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). https://www.tensorflow.org/ Software available from tensorflow.org.
2. Jacques Bertin. 1983. Semiology of graphics: diagrams, networks, maps. (1983).
3. Wray L Buntine. 1994. Operations for learning with graphical models. *Journal of artificial intelligence research* 2 (1994), 159–225.
4. Michael Gleicher. 2018. Considerations for Visualizing Comparison. *IEEE transactions on visualization and computer graphics* 24, 1 (2018), 413–423.
5. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
6. Minsuk Kahng, Pierre Y Andrews, Aditya Kalro, and Duen Horng Polo Chau. 2018. ActiVis: Visual



	.61	.61	.58	.56	.54	.52	.50	.48	.45	.44	.44	.43	.43
Created At	23:44	20:57	20:01	20:56	23:50	18:58	23:50	23:47	20:59	20:01	21:15	21:17	20:29
Val Acc	.61	.61	.58	.56	.54	.52	.50	.48	.45	.44	.44	.43	.43
Num Epochs	2	20	27	20	4	3	2	4	20	27	20	20	20
Num Params	187.1m	871.6k	406.5k	546.7k	418.6b	105.9k	46.5m	3.6t	251.7m	789.0k	2.0m	3.4m	789.0k
Forward Time(s)	0.0	3.6	1.1	2.3	0.0	0.7	0.0	0.0	3.8	0.6	3.0	3.3	1.6
Backward Time(s)	-1.0	5.0	4.0	5.0	-1.0	5.0	-1.0	-1.0	5.0	4.0	5.0	5.0	5.0
Val Acc / log(Params)	0.07	0.10	0.10	0.10	0.05	0.10	0.06	0.04	0.05	0.08	0.07	0.07	0.07
Val Acc / log(Fwd)	0.00	0.17	0.19	0.17	0.00	0.18	0.00	0.00	0.13	0.16	0.13	0.12	0.14
Val Acc / log(Back)	0.00	0.88	0.97	0.80	0.00	0.75	0.00	0.00	0.65	0.74	0.62	0.62	0.62

Figure 5. Three examples of SNACs being used in visual analytics applications, each described in the text. SNACs are space-efficient and versatile, and can fit the space and positioning constraints required by systems that help users compare architectures.

Exploration of Industry-Scale Deep Neural Network Models. *IEEE transactions on visualization and computer graphics* 24, 1 (2018), 88–97.

7. Jan Koutník, Jürgen Schmidhuber, and Faustino Gomez. 2014. Evolving deep unsupervised convolutional networks for vision-based reinforcement learning. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. ACM, 541–548.

8. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

9. Yann LeCun, Yoshua Bengio, and others. 1995. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks* 3361, 10 (1995), 1995.

10. Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, and others. 2015. Going deeper with convolutions. Cvpr.

11. F-Y Tzeng and K-L Ma. 2005. Opening the black box-data driven visualization of neural networks. In *Visualization, 2005. VIS 05. IEEE*. IEEE, 383–390.

12. Kanit Wongsuphasawat, Daniel Smilkov, James Wexler, Jimbo Wilson, Dandelion Mané, Doug Fritz, Dilip Krishnan, Fernanda B Viégas, and Martin Wattenberg. 2018. Visualizing Dataflow Graphs of Deep Learning Models in TensorFlow. *IEEE transactions on visualization and computer graphics* 24, 1 (2018), 1–12.

13. Matthew D Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *European conference on computer vision*. Springer, 818–833.