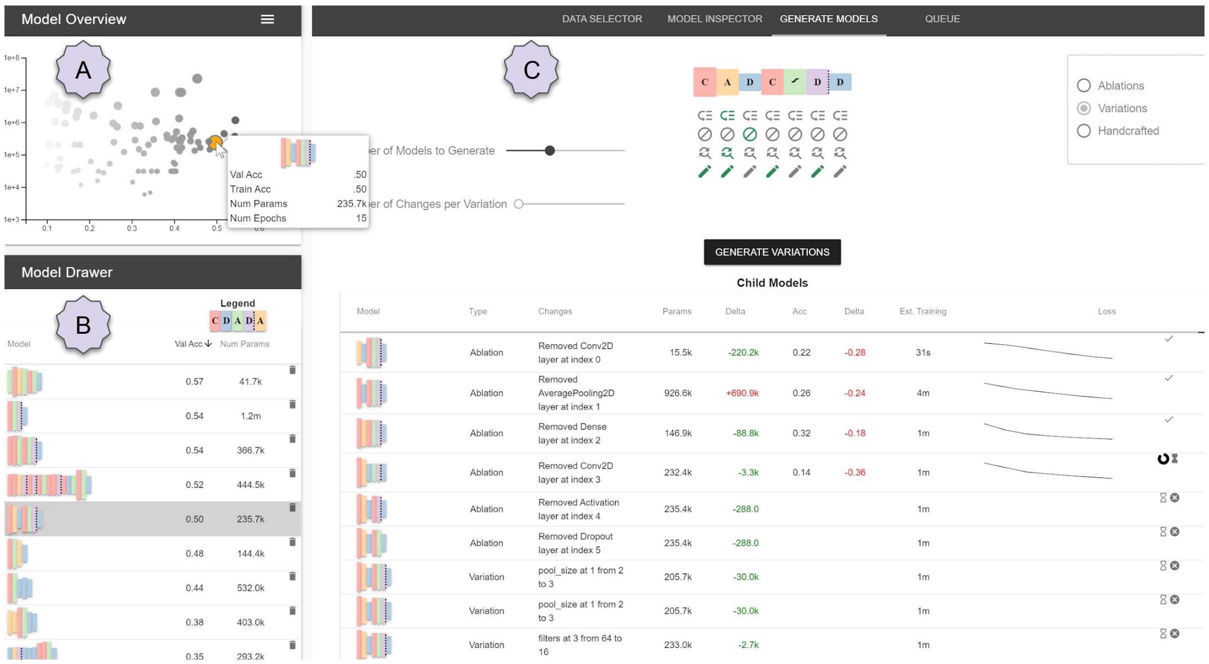
Ablate, Variate, and Contemplate: Visual Analytics for Discovering Neural Architectures

Dylan Cashman, Adam Perer, Remco Chang, Hendrik Strobelt



Val Acc	.50
Train Acc	.50
Num Params	235.7k
Num Epochs	15

Model	Val Acc↓	Num Params
	0.57	41.7k
	0.54	1.2m
	0.54	366.7k
	0.52	444.5k
	0.50	235.7k
	0.48	144.4k
	0.44	532.0k
	0.38	403.0k
	0.35	293.2k

Child Models

Model	Type	Changes	Params	Delta	Acc	Delta	Est. Training	Loss
	Ablation	Removed Conv2D layer at index 0	15.5k	-220.2k	0.22	-0.28	31s	✓
	Ablation	Removed AveragePooling2D layer at index 1	926.6k	+690.9k	0.26	-0.24	4m	✓
	Ablation	Removed Dense layer at index 2	146.9k	-88.8k	0.32	-0.18	1m	✓
	Ablation	Removed Conv2D layer at index 3	232.4k	-3.3k	0.14	-0.36	1m	
	Ablation	Removed Activation layer at index 4	235.4k	-288.0			1m	
	Ablation	Removed Dropout layer at index 5	235.4k	-288.0			1m	
	Variation	pool_size at 1 from 2 to 3	205.7k	-30.0k			1m	
	Variation	pool_size at 1 from 2 to 3	205.7k	-30.0k			1m	
	Variation	filters at 3 from 64 to 16	233.0k	-2.7k			1m	

Fig. 1: A screenshot of the REMAP system. In the Model Overview, section A, a visual overview of the set of sampled models is shown. Darkness of circles encodes performance of the models, and radius encodes the number of parameters. In the Model Drawer, section B, users can save models during their exploration for comparison or to return to later. In section C, four tabs help the user explore the model space and generate new models. The Generate Models tab, currently selected, allows for users to create new models via ablations, variations, or handcrafted templates.

Abstract— The performance of deep learning models is dependent on the precise configuration of many layers and parameters. However, there are currently few systematic guidelines for how to configure a successful model. This means model builders often have to experiment with different configurations by manually programming different architectures (which is tedious and time consuming) or rely on purely automated approaches to generate and train the architectures (which is expensive). In this paper, we present Rapid Exploration of Model Architectures and Parameters, or REMAP, a visual analytics tool that allows a model builder to discover a deep learning model quickly via exploration and rapid experimentation of neural network architectures. In REMAP, the user explores the large and complex parameter space for neural network architectures using a combination of global inspection and local experimentation. Through a visual overview of a set of models, the user identifies interesting clusters of architectures. Based on their findings, the user can run ablation and variation experiments to identify the effects of adding, removing, or replacing layers in a given architecture and generate new models accordingly. They can also handcraft new models using a simple graphical interface. As a result, a model builder can build deep learning models quickly, efficiently, and without manual programming. We inform the design of REMAP through a design study with four deep learning model builders. Through a use case, we demonstrate that REMAP allows users to discover performant neural network architectures efficiently using visual exploration and user-defined semi-automated searches through the model space.

Index Terms—visual analytics, neural networks, parameter space exploration

✦

1 Introduction

- *Dylan Cashman and Remco Chang are with Tufts University, USA E-mail: dcashm01@cs.tufts.edu, remco@cs.tufts.edu*
- *Adam Perer is with Carnegie Mellon University, USA E-mail: adamperer@cmu.edu*
- *Hendrik Strobelt is with the MIT IBM Watson AI Lab, USA E-mail: hendrik.strobelt@ibm.com*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

Deep neural networks have been applied very successfully in recent advances in computer vision, natural language processing, machine translation and many other domains. However, in order to obtain good performance, model developers must configure many layers and parameters carefully. Issues with such manual configuration have been raised as early as 1989, where Miller et al. [40] suggested automated neural architecture search should be useful in enabling a wider audience to use neural networks:

"Designing neural networks is hard for humans. Even small networks can behave in ways that defy comprehension; large, multi-layer, nonlinear networks can be downright mystifying." [40]

Thirty years later, the authors' note is still a common refrain. While research has continued in automated neural architecture search, much of the progress in algorithms has focused on developing more performant models using prohibitively expensive resources. For example, state of the art algorithms in reinforcement learning taking 1800 GPU days [73] and evolutionary algorithms taking 3150 GPU days [48] to discover their reported architectures. Those users that have access to the type of hardware necessary to use these algorithms likely would either have the expertise needed to manually construct their own network or would have access to a machine learning expert that would be able to do it for them.

Likewise, a number of visual analytics tools have been released that make neural networks more interpretable and customizable [18]. However, these tools presuppose that a sufficiently performant model architecture has been chosen *a priori* without the aid of a visual analytics tool. The initial choice of neural network architecture is still a significant barrier to access that limits the usability of neural networks. Tools are needed to provide a human-driven search for neural network architectures to provide a data scientist with an initial performant model. Once this model has been found, existing visual analytics tools could be used to fine tune it, if needed.

In this work, we present REMAP, a tool for human-in-the-loop neural architecture search. Compared to the manual discovery of neural architectures (which is tedious and time consuming), REMAP allows a model builder to discover a deep learning model quickly via exploration and rapid experimentation. In contrast to fully automated algorithms for architecture search (which are expensive and difficult to control), REMAP uses a semi-automated approach where users have fine-grained control over the types of models that are generated. This allows users to trade off between the size of the model, the performance on individual classes, and the overall performance of the resulting model.

Through a set of interviews with model builders, we establish a set of tasks used in the manual discovery of neural network architectures. After developing an initial version of REMAP, we held a validation study with the same experts and incorporated their feedback into the tool. In REMAP, users first explore an overview of a set of pre-trained small models to find interesting clusters of models. Then, users guide the discovery of new models via two operations on existing models: ablations, in which a new model is generated by removing a single layer of an existing model, and variations, in which several new models are generated by random atomic changes of an existing model, such as a reparameterization or the replacement of an existing layer. Users can also manually construct or modify any architecture via a simple drag-and-drop interface. By enabling global and local inspection of networks and allowing for user-directed exploration of the model space, REMAP supports model selection of neural network architectures for data scientists.

The model space for neural networks poses unique challenges for our tool. Whereas many of the parameter spaces explored in other types of models have a set number of choices of parameters, the parameter space for neural networks is potentially infinite - one can always choose to add more layers to a network. In order to aid in the interpretation of the model space, we propose 2-D projections based on two different distance metrics for embedding neural networks based on Lipton's two forms of model interpretability, *transparency* and *post-hoc interpretability* [34].

The second significant hurdle for a visual model selection over neural networks is to find a visual encoding for neural networks that enabled comparison of many networks while still conveying shape and computation of those networks. In this work, we contribute a novel visual encoding, called Sequential Neural Architecture Chips (SNACs), which are a space-efficient, adaptable encoding for feed-forward neural networks. SNACs can be incorporated into both visual analytics systems and static documents such as academic papers and industry white papers.

The workflow of our system largely follows the conceptual framework for visual parameter space analysis from Sedlmair et. al. [51]. A starting set of models is initially sampled from the space in a preprocessing stage, and projections of the models are calculated. Models are then explored in three derived spaces: two MDS projections corresponding to the two distance metrics as well as a third projection with interpretable axes. The system then uses the *global-to-local* strategy of navigating the parameter space, moving from an overview of models to an inspection of individual models in neighborhoods in the derived spaces. During exploration, users can instruct the system to spawn additional models in the neighborhood of already-sampled models, rendering more definition in their mental model of the parameter space on the regions they are most interested in.

Overall, the contributions of this paper include:

- REMAP, a visual analytics system for semi-automated neural architecture search that is more efficient than existing manual or fully-automated approaches
- A set of visual encodings and embedding techniques for visualizing and comparing a large number of sequential neural network architectures
- A set of design goals derived from a design study with four model builders
- A use case applying REMAP to discover convolutional neural networks for classification of sketches

2 Motivation

A machine learning *model* is an algorithm that predicts a target label from a set of predictor variables. These models learn how to make their prediction by learning the relationships between the predictor variables and target label on a *training dataset*. Machine learning models typically train by iterating over the training set multiple times; each iteration is called an *epoch*. In each epoch, the model makes predictions and accrues *loss* when it makes poor predictions. It then updates its learned parameters based on that loss. At each epoch, the accuracy of the model on a held out portion of the dataset, called the *validation dataset*, is calculated.

Neural networks are a class of machine learning models that are inspired by the message passing mechanisms found between neurons in brains. A neural network consists of an architecture and corresponding parameters[1] chosen by the model builder for each component of that architecture. The architecture defines the computational graph mapping from input to output, e.g. how the input space, such as an image, is transformed into the output space, such as a classification (the image is a *cat* or a *dog*). In sequential neural networks, which have simple computation graphs representable by linked lists, the nodes of the computations graphs are called *layers*.

Choosing an architecture that performs well can be difficult [40]. Small changes in parameters chosen by model builders can result in large changes in performance, and many configurations will result in models that quickly plateau without gaining much predictive capacity through training. In addition, training neural networks is very slow relative to other machine learning methods. As a result, the process of manually discovering a performant model can be frustrating and costly in time and resources.

Automated algorithms for neural architecture search generate thousands of architectures in order to find performant architectures [72] and can require tens of thousands of GPU hours of training [48, 73]. The best discovered models might be too large for a model builder if they aim to deploy their model on an edge device such as a tablet or an internet of things device. Ideally, a model builder would be able to handcraft each generated model and monitor its training to not waste time and resources discovering models that were not useful. However, handcrafting each model can be time consuming and repetitive.

[1]Parameters chosen by the model builder are sometimes called hyperparameters to differentiate from the parameters of a model that are learned during training. In this work, we call both of these terms parameters, but refer to the latter as learned parameters for the sake of delineation.

In our tool, we seek a middle ground. We initially sample a small set of architectures, and then use visualizations to facilitate exploration of the model space. Model builders can find regions of the space that produce models they are interested in, and then they can execute a local, constrained, automated search near those models. As they get closer to finding an acceptable model, they can explicitly handcraft models through a graphical interface. Rather than training thousands of architectures, the model builder trains orders of magnitude less, and stops the architecture search when they have found an acceptable model. Our semi-automated approach lets the user search for neural architectures without the tedium of manually constructing each model and without the resources and time required by fully-automated algorithms for neural architecture search.

3 Related Work

3.1 Neural Architecture Search

Algorithms for the automated discovery of neural network architectures were proposed as early as the late 1980s using genetic algorithms [40]. Algorithm designers were concerned that neural networks were excessively hard to implement due to their large parameter space and odd reaction to poor parameterizations. In recent years, interest in neural networks has exploded as they have proven to be state of the art algorithms for image classification [29], text classification [31], video classification [25], image captioning [67], visual question answering [39], and a host of other classic artificial intelligence problems. An increased interest in automated neural architecture searches has followed, resulting in a variety of algorithms using Bayesian optimization [56], network morphisms [20], or reinforcement learning [4,72]. These algorithms typically define the architecture space so that it is easily searchable by classical parameter space exploration techniques, such as gradient-based optimization [24,35]. Elsken et al. provide a summary of new research in algorithmic methods in a recent survey [12].

Such methods are driven by an attempt to compete with state of the art performant architectures such as ResNet [17] or VGGNet [54] that were carefully handcrafted based on years of incremental research in the community. Because performance has been the primary motivator, automated neural architecture search algorithm designers have depended on expensive hardware setups using multiple expensive GPUs and very long search and training times [35]. As a result, the use of these algorithms is out of reach for many potential users without expensive hardware purchases or large outlays to cloud machine learning services. In contrast, our tool is more accessible to data scientists because it drastically shrinks the search space by conducting user-defined local, constrained searches in neighborhoods around models the user is interested in.

3.2 Visualization for Neural Networks

Visualization has been used in both the machine learning literature and the visual analytics literature for understanding and diagnosing neural networks. In particular, attempts have been made to explain the decision making process of trained networks. Saliency maps [53] and gradient-based methods [52] were an early attempt to understand which pixels were most salient to a network's predictions in image classification networks. However, recent work has shown that saliency maps may be dependent only on inherent aspects of the image and not the network's decision making, calling into doubt some of the truthfulness of such methods [2]. Methods also exist which inspect the effect of individual layers on the decisions of the network [68,69]. *Lucid* is a library built on the *Tensorflow* machine learning library for generating various visualizations of networks [45].

Visual analytics tools extend these techniques by offering interactive environments for users to explore their networks. Some tools allow users to inspect how various components of a trained network contribute to its predictions [21, 37, 58, 65, 66], while others allow the user to build and train toy models to understand the influence of various hyperparameter choices [23,55] Other tools focus on debugging a network to determine which changes must be made to improve its performance by viewing the activations, gradients, and failure cases of the network [7,36,47,57]. Hohman et al. provide a comprehensive overview of visual analytics for deep learning [18] .

All of these visual analytics tools presuppose that the user has selected an architecture and wants to inspect, explain, or diagnose it. In contrast, REMAP allows the user to discover a new architecture. A user of REMAP might take the discovered architecture and then feed it into a tool such as DeepEyes to more acutely fine tune it for maximal performance [47].

3.3 Visual Analytics for Model Selection

Model selection is highly dependent on the needs of the user and the deployment scenario of a model. Interactivity can be helpful in comparing multiple models and their predictions on a holdout set of data. Zhang et. al. recently developed Manifold, a framework for interpreting machine learning models that allowed for pairwise comparisons of various models on the same validation data [70]. Mühlbacher and Piringer support analyzing and comparing regression models based on visualization of feature dependencies and model residuals [42]. Schneider et al. demonstrate how the visual integration of the data and the model space can help users select relevant classifiers to form an ensemble [50]. Snowcat is a visual analytics tool that enables model selection from a set of black box models returned from a automated machine learning backend by visually comparing their predictions in the context of the data source [6]. These methods all assume that the model is being selected from a set of pretrained models, in contrast to our system which can generate additional models based on user input.

3.4 Visual Analytics for autoML

Automated Machine Learning, or autoML, comprises a set of techniques designed to automate the end-to-end process of ML. To accomplish this, autoML techniques automate a range of ML operations, including but not limited to, data cleaning, data pre-processing, feature engineering, feature selection, algorithm selection and hyperparameter optimization [16]. Different autoML libraries such as AutoWeka [27,61], Hyperopt [5,26], and Google Cloud AutoML [33] are in use either commercially or as open source tools.

Visual Analytics systems have been used to both provide an interface to the autoML process as well as insert a human in the loop of various parts of the process. TreePOD [41] helps users balance potentially conflicting objectives such as accuracy and interpretability of automatically generated decision tree models by facilitating comparison of candidate tree models. Users can then spawn similar decision trees by providing variation parameters, such as tree depth and rule inclusion. BEAMES [11] allows users to search for regression models by offering feedback on an initial set of models and their predictions on a held out validation dataset. The system spawns new models based on that feedback, and users iterate until they find a satisfactory model. Various tools facilitate user control over the generation of models for regression [42], clustering [8,30,43,49], classification [9,64], dimension reduction [3, 10, 19, 38, 44]. REMAP differs from those tools in that it explicitly uses properties of neural networks, such as the sequence of layers, in its visual encodings. Also, because neural networks take much longer to train than decision trees, regression models, and most models considered by previous visual analytics tools, REMAP places more of an emphasis on only generating models that the user is interested in.

4 Design Study

In order to develop a set of task requirements, we interviewed a set of model architects about their practices in manually searching for neural network architectures. We also asked the experts what visualizations might be helpful for non-experts in a human-in-the-loop system for neural network architecture search.

Participants: To gather participants, we recruited individuals with experience in designing deep neural network architectures. Four experienced model builders agreed to participate in the interview study. Three of the participants are PhD students in machine learning, and the fourth participant has a Masters degree in Computational Data Science and works in industry. They had previously used neural networks for medical image classification, image segmentation, natural language processing, and graph inference. One participant contributed to an open source automated neural architecture search library. All four participants were from different universities or companies and had no role in this project. Participants were compensated with a twenty dollar gift card.

Method: Interviews were held with each participant to establish a set of tasks used to manually discover and tune neural networks. The interviews were held one-on-one using an online conferencing software with an author of this work and took one hour each. Audio was recorded and transcribed with the participants' consents so that quotes could be taken.

Interviews were semi-structured, with each participant being asked the same set of open-ended questions[2]. They were first asked to describe their work with neural networks, including what types of data they had worked with. They were then asked about their typical workflow in choosing and fine tuning a model. Then, the benefits of human-in-the-loop systems for neural network model selection were discussed. Lastly, participants were prompted for what types of features might be useful in a visual analytics system for selecting a neural network.

Findings: The findings from the interview study resulted in the following set of design goals.

- **Goal G1: Find Baseline Model:** Three out of the four participants noted that when they are building an architecture for a new dataset, they start with a network that they know is performant. This network might be from a previous work in the literature or it might be a network they've used for a different dataset. This network typically provides a baseline, upon which they then do fine tuning experiments: "*The first step is just use a structure proposed in the paper. Second step I always do is to change hyperparameters. For example, I add another layer or use different dropouts.*" One participant noted that they prioritize using a small model as a baseline because they are more confident in the stability of small models, and it is easier to run fine tuning experiments on small models because they train faster.
- **Goal G2: Generate Ablations and Variations:** Three participants noted that in order to drive their fine tuning, they typically do two types of experiments on a performant network. First, they do ablation studies, a technical term referring to a set of controlled experiments in which one independent variable is turned off for each run of the experiment. Based on the results of the ablation studies, they then generate variations of the architecture by switching out or reparameterizing layers that were shown to be less useful by the ablations. Two participants noted that these studies can be onerous to run, since they need to write code for each version of the architecture they try.
- **Goal G3: Explain/Understand Architectures:** When asked about the types of information to visualize for data scientists, two participants noted that users might be able to glean a better understanding of how neural networks are constructed by viewing the generated architectures. While it may be obvious to the study participants that convolutional layers early in the network are good at extracting features but less helpful in later layers, that understanding comes from experience. By visually comparing models, non-experts might come to similar conclusions. One participant pointed out that the human-in-the-loop could interpret the resulting model more, helping "*two people, the person developing the results, and the person buying the algorithm.*"
- **Goal G4: Human-supplied Constrained Search:** Participants were asked what role a human-in-the-loop would have in selecting a neural network architecture, compared to a fully-automated model search. All four participants noted that if the data is clean and correctly labeled, and there are sufficient resources and time, that a human-in-the-loop would not improve upon an automated neural architecture search. But three participants noted that when resources are limited, the human user can compensate by offering constraints to an automated search, pointing an automated search to particular parts of the model space that are more interesting to the user. One participant noted that for fully-automated model search, "*some use reinforcement learning, [some] use Bayesian optimization. The human can also be the controller.*"

[2]Interview questions are available as a supplemental document.

From these findings, we distill the following tasks that our system must support to enable data scientists to discover performant neural network architectures.

- **Task T1: Quickly search for baseline architectures through an overview of models.** Users must be able to start from an effective baseline architecture [G1]. Experts typically refer to the literature to find a starting architecture that has already been shown to work on a similar problem, such as VGGNet [54] or ResNet [17]. These models, however, have hundreds of millions of parameters and cannot be easily and quickly experimented upon, so some other manner for finding compact, easily trainable baseline models is needed. Users should be able to find small, performant baseline models easily via visual exploration.
- **Task T2: Generate local, constrained searches in the neighborhood of baseline models.** Our tool needs to provide the ability to explore and experiment on baseline models using ablations and variations [G2]. These experiments should help the user in identifying superfluous layers in an architecture. The human user should be able to provide simple constraints to the search for new architectures [G4].
- **Task T3: Visually compare subsets of models to understand small, local differences in architecture.** The tool should support visual comparisons of models to help the user understand what components make a successful neural network architecture. This helps the user interpret the discovered neural network models [G3] while also informing the user's strategies for generating variations and exploring the model space [G4].

Beyond these three tasks, we also note that compared to many fully automated neural architecture searches, we must be cognizant of limitations on resources. Much of the neural network literature assumes access to prohibitively expensive hardware and expects the user to wait hours or days for a model to train. In our tool, we focus instead on small models that are trainable on more typical hardware. While these models may not be state of the art, they are accessible to a much wider audience.

5 REMAP: Rapid Exploration of Model Architectures and Parameters

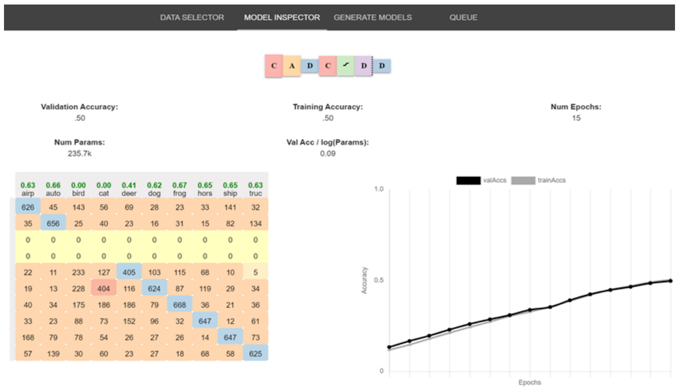
REMAP is a client-server application that enables users to interactively explore and discover neural network architectures.[3] A screenshot of the tool can be seen in Figure 1. The interface features three components: a Model Overview represented by a scatter plot (Fig. 1A), a Model Drawer for retaining a subset of interesting models during analysis (Fig. 1B), and a data/model inspection panel (Fig. 1C).

All screenshots in this section use the CIFAR-10 dataset, a collection of 50,000 training images and 10,000 testing images each labeled as one of ten mutually exclusive classes [28]. Model training including both preprocessing and in-situ model generation was done using a Dell XPS 15 laptop with a 2.2ghz i7-8750 processor, 32 GB of RAM, and a NVIDIA GeForce GTX 1050 Ti GPU with 4GB of VRAM.

5.1 General Workflow

The user workflow for REMAP is inspired by the common workflow identified in the interview study and encompasses tasks T1, T2, and T3 as defined in section 4. First, they find a baseline model by visually exploring a set of pre-trained models in the Model Overview [T1], seen in Figure 1A. They select models of interest by clicking on their respective circles, placing them into the Model Drawer, seen in Figure 1B. By mousing over models in the overview and scanning the Model Drawer, users can visually compare models of interest [T2]. Then, they use the ablation and variation tools [T3] to fine tune each model of interest,

[3]The source code for the tool along with installation instructions are publicly available at `https://github.com/dylancashman/remap_nas`.



Validation Accuracy: .50

Training Accuracy: .50

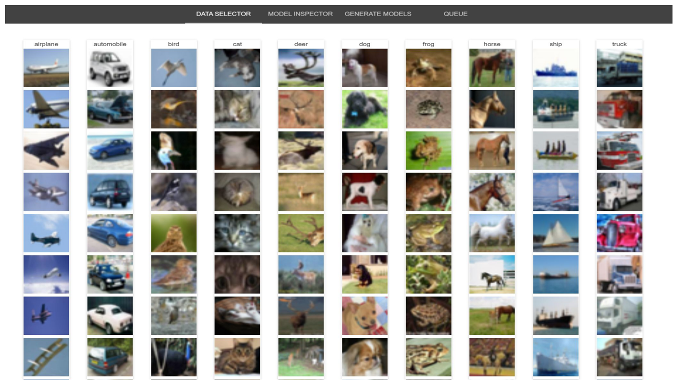
Num Epochs: 15

Num Params: 235.7k

Val Acc / log(Params): 0.09

0.63 airp	0.66 auto	0.00 bird	0.00 cat	0.41 deer	0.62 dog	0.67 frog	0.65 hors	0.65 ship	0.63 truc
626	45	143	56	69	28	23	33	141	32
35	656	25	40	23	16	31	15	82	134
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
22	11	233	127	405	103	115	68	10	5
19	13	228	404	116	624	87	119	29	34
40	34	175	186	186	79	668	36	21	36
33	23	88	73	152	96	32	647	12	61
168	79	78	54	26	27	26	14	647	73
57	139	30	60	23	27	18	68	58	625

(a)



(b)

Fig. 2: (a) The model inspection tab lets users see more granular information about a highlighted model. This includes a confusion matrix showing which classes the model performs best on or misclassifies most frequently. Users can also view training curves to determine if an architecture might be able to continue to improve if trained further. (b) By selecting individual classes from the validation data, users can update the darkness of circles in the the Model Overview to see how all models perform on a given class.

as seen in Figure 1C. These tools spawn new models with slightly modified architectures that train in the background, which in turn get embedded in the Model Overview. Instructions for new models are sent back to the server. The server maintains a queue of models to train and communicates its status after each epoch of training.

Users iterate between exploring the model space to find interesting baseline models and generating new architectures from those baseline models. For the types of small models explored in this tool, training can take 1-3 minutes for a single model. Users can view the current training progress of child models in the Generate Models tab, or can view the history of all training across all models in the Queue tab. In the Queue tab, they can also reorder or cancel models if they don't want to wait for all spawned models to train.

If users are particularly interested in performance on certain classes in the data, they can select a data class using the Data Selector seen in Figure 2b to modify the Model Overview. Users can also see a confusion matrix corresponding to each model in the Model Inspector tab, seen in Figure 2a. By interacting with both the model space and the data space, they are able to find models that match their understanding of the data and the importance of particular classes.

5.2 Preprocessing

In order to provide a set of model baselines, REMAP must generate a set of initial models. This set should be diverse in the model space, using many different combinations of layers in order to hopefully cover the space. That way, whether the user hopes to find a model that performs well on a particular class or that has a particularly small number of parameters, there will exist a reasonable starting point to their model search.

REMAP generates this initial model space by using a random scheme based on automated neural architecture searches in the literature [12]. A Markov Chain is defined which dictates the potential transition probabilities from layer to layer in a newly sampled model. Starting from an initial state, the first layer is sampled, then its hyperparameters are sampled from a grid. Then, its succeeding layer is sampled based on what valid transitions are available. Transition probabilities and layer hyperparameters were chosen based on similar schemes in the autoML literature [4], as well as conventional rules of thumb. For example, convolutional layers should not follow dense layers because the dense layers remove the locality that convolutional layers depend on. In essence, REMAP uses a small portion of a random automated neural architecture search to initialize the human-in-the-loop search. For models in this section and in screenshots, 100 initial models were generated and trained for 10 epochs each, taking approximately 4 hours. While that is a nontrivial amount of required preprocessing time, it compares favorably to the tens of thousands of GPU hours required by a fully automated search [48, 73], which might sample over 10,000 models [72].

5.3 Model Overview

The top left of the interface features the Model Overview (Fig. 1A), a scatter plot which visualizes three different 2D projections of the set of models. The user is able to toggle between the different 2D projections. The visual overview of the model space serves two purposes. First, it can serve as the starting point for model search, where users can find small, performant baseline models to further analyze and improve. The default view plots models on interpretable axes of validation accuracy vs. a log scale of the number of parameters, visible in Figure 1. Each circle represents a trained neural network architecture. The darkness of the circle encodes the accuracy of the architecture on a held out dataset, with darker circles corresponding to better accuracy. The radius of the circle encodes the log of the number of parameters. This means that in the default projection, the validation accuracy and the number of parameters are double encoded - this is based on the finding from the interview study that finding a small, performant baseline model is the first step in model selection. The lower right edge of the scatter plot forms a Pareto front, where model builders can trade off between performance of a model and its size, similar to the complexity vs. accuracy plots found in Muhlbacher et al.'s TreePOD tool for decision trees [41].

Once baseline models have been selected, the Model Overview can also be used to facilitate comparisons with neighbors of the baseline. Users are able to view details of neighboring architectures by hovering over their corresponding points in the overview. By mousing around a neighborhood of an interesting baseline model, they might be able to see how small changes in architecture affect model performance. However, it is well known that neural networks are notoriously fickle to small changes in parameterization [40]. Two points close together in that view could have wildly different architectures.

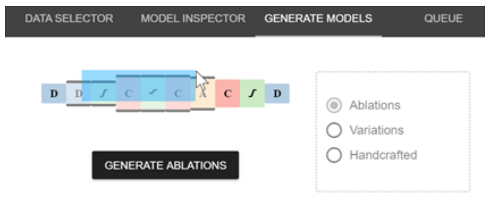
To address this, REMAP offers two additional projections based on two distance metrics between neural networks. The two metrics are based on the two types of model interpretability identified in Lipton's recent work [34]: structural and post-hoc. Their respective projections are seen in Figure 4b, with the same model highlighted in orange in both projections. 2-D Projections are generated from distance metrics using `scikit-learn`'s implementation of Multidimensional Scaling [46].

Structural interpretability refers to the interpretability of how the components of a model function. A distance metric based on structural interpretability would place models with similar computational components, or layers, close to each other in the projection. We used OTMANN distance, an Optimal Transport-based distance metric that measures how difficult it is to transform one network into another, similar to the Wasserstein distance between probability distributions [24]. The resulting projection is seen in section B of Figure 4b. Projecting by this metric allows users to see how similar architectures can result in large variances in validation accuracy and number of parameters.

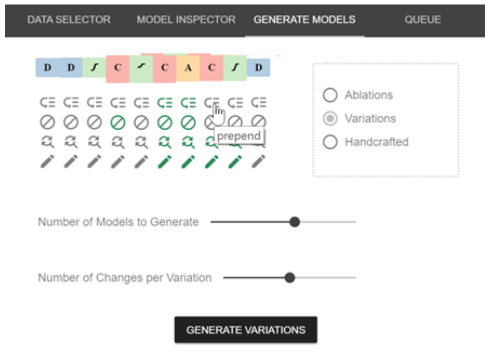
Post-hoc interpretability refers to understanding a model based on its predictions. A distance metric based on post-hoc interpretability would place models close together in the projection if they have similar predictions on a held-out test set. Ideally, this notion of similarity should be more sophisticated than simply comparing their accuracy on the entire test set — it should capture if they usually predict the same even on examples that they classify incorrectly. We use the edit distance between the two architectures' predictions on the test set. The resulting projection is seen in section C of Figure 4b. It can be used to find alternative baseline architectures that have similar performance to models of interest.

New models generated via ablations and variations are embedded in the Model Overviews via an out-of-sample MDS algorithm [62]. Users can view how spawned models differ from their parent models in the different spaces and get a quick illustration of which qualities were inherited by the parent model.

5.4 Ablations and Variations



(a)



(b)

Fig. 3: Controls for creating (a) Ablations and (b) Variations. Users toggle between the two types of model generation with a radio button. Ablations create a set of models, one for each layer with that layer removed, to communicate the importance of each layer. The Variations feature runs constrained searches in the neighborhood of a selected model. Users toggle which types of variations are allowed for each layer, as well as the number of variations allowed per model

According to our expert interviews, an integral task in finding a performant neural network architecture is to run various experiments on slightly modified versions of a baseline architecture. One type of modification that is done is an ablation study, in which the network is retrained with each feature of interested turned off, one at a time. The goal of ablations is to determine the effect of each feature of a network. This might then drive certain features to be pruned, or for those features to be duplicated.

In our system, users can automatically run ablation studies that retrain a selected model without each of its layers. The system will then train those models for the same number of epochs as the parent model, and display to the user the change in validation accuracy. If the user wants to make a more fine-grained comparison between the models, the user can move the model resulting from an ablation into the Model Drawer, and then use the Model Inspector to compare their confusion matrices.

Using the Variations feature in REMAP, seen in Figure 3b, users can sample new models that are similar to the baseline model. By default, the variation command will randomly remove, add, replace, or reparameterize layers. Users can constrain the random generation of variations by specifying a subset of types of variations for a given layer. For example, a user might not want to remove or replace a layer that was very important according to the ablation studies, but could still allow it to be reparameterized. Valid variation types are *prepend* with a new layer, *remove* a layer, *replace* a layer, or *reparameterize* a layer.

When generating ablations and variations, the user is shown each child model generated from the baseline model that is selected (Fig. 1C). Changes that were made to generate that model are shown as well. By viewing all children on the same table, the user may be able to see the effect of certain types of changes; e.g. adding a dense layer typically dramatically increases the number of parameters, while adding a convolutional layer early sometimes increases the validation accuracy. Spark lines communicate the loss curve of each child model as it trains. Each child model is embedded into the Model Overview, and can be moved to the Model Drawer to become a model baseline.

5.5 Sequential Neural Architecture Chips

We developed a visual encoding, SNAC (Sequential Neural Architecture Chip), for displaying sequential neural network architectures. Seen in Figure 4a, SNAC is designed to facilitate easy visual comparisons across several architectures via juxtaposition in a tabular format. Popular visual encodings used in the machine learning [17, 29, 32, 60, 69] and visual analytics literature [22, 63, 66] take up too much space to fit multiple networks on the same page. In addition, the layout of different computational components and the edges between them makes comparison via juxtaposition difficult [15].

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 displayed in most other visualizations of networks. Layer type is redundantly encoded with both color and symbol. 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 sigmoid. Dropout layers feature a dotted border to signify that some activations are being dropped. The height of each block corresponds to the data size on a bounded log scale, to indicate to the user whether the layer is increasing or decreasing the dimensionality of the activations flowing through it. SNACs are available as an open source component for use in publications and visual analytics tools.[4].

6 Expert Validation Study

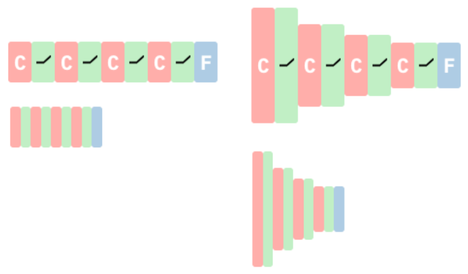
The initial version of REMAP was developed based on a design study described in section 4. Two months later, a validation study was held with the same four model builders that participated in the design study. The goal of the validation study was to assess whether the features of REMAP were appropriate and sufficient to enable a semi-automated model search, and to determine if the system aligned with the mental model of deep learning model builders. Users were asked to complete two tasks using REMAP, and then provide feedback on how individual features supported them in their tasks.

Participants: The same four individuals with experience in designing deep neural network architectures that participated in the first study agreed to participate in the validation study. Participants were compensated with a forty dollar gift card.

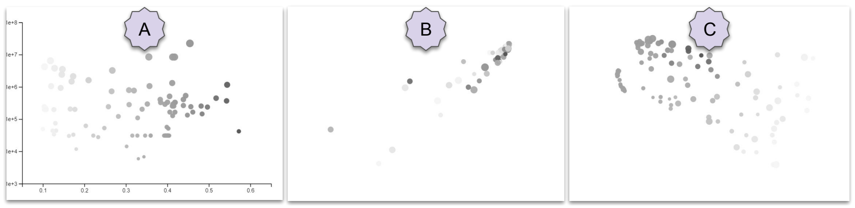
Method: Interviews were again held one-on-one using an online conferencing software and took approximately two hours each. Audio of the conversation as well as screen sharing were recorded.

At the start of the study, participants were first given a short demo of the system, with the interviewer sharing their screen and demonstrating all of the features of REMAP. Then, participants were given access

[4]The open source implementation of SNACs can be viewed at `http://www.eecs.tufts.edu/~dcashm01/snacs/`



(a)



(b)

Fig. 4: (a) The *SNAC* visual encoding of a neural network architectures, seen at four different resolutions. This architecture has a three convolutions, each followed by an activation, and concludes with a fully connected layer. (b) Three alternative visual overviews of the model space. Section A shows the set of models on a set of interpretable axes, validation accuracy vs. log of the number of parameters. Sections B and C use multidimensional scaling to lay out the same set of models based on structural similarity (B) and prediction similarity (C). The darkness of the circle encodes the model accuracy, and the radius of the circle encodes the log of the number of parameters.

to the application through their browser and were given two tasks to complete using the tool. The participant's screen was recorded during their completion of the two tasks. Participants were asked to evaluate the features of the tool through their usage in completing their tasks. One of the four participants was unable to access the application remotely, and as a result, directed the interviewer on what interactions to make in REMAP and followed along as the interviewer shared their screen.

Both tasks consisted of discovering a performant neural network architecture for image classification on the CIFAR-10 dataset, a collection of 50,000 training images and 10,000 testing images each labeled as one of ten mutually exclusive classes [28]. This dataset was chosen because all four experts had experience building neural network architectures for this dataset. This allowed the participants to quickly assess whether the system enabled them to do the types of operations they might have done manually searching for an architecture on CIFAR-10. In this evaluation, we report participants' feedback on whether the tool enabled them to navigate the model space in a similar manner to their manual model discovery process.

Tasks: The first task given to the participants was to simply find the neural network architecture that would attain the highest accuracy on the 10,000 testing images of CIFAR-10. For the second task, participants were given a scenario that dictated constraints on the architecture they had to find. Participants were asked to find a neural network architecture for use in a mobile application used by bird watchers in a certain park that had many birds and many cats. Birds and Cats are two of the ten possible labels in the CIFAR-10 dataset. The resulting architecture needed to prioritize high accuracy on those two labels, and also needed to have under 100,000 parameters so that it would be easily deployable on a mobile phone. The two tasks were chosen to emulate two types of usage for REMAP: unconstrained model search and constrained model search.

Participants were given up to an hour to complete the two tasks and were encouraged to ask questions and describe their thought process. Then, they were asked about the efficacy of each feature in the tool.

Findings: Participants were able to select models for both tasks. However, each participant expressed frustration at the lack of fine-grained control over the model building process. In general, participants found that the tool could be useful as an educational tool for non-experts because of the visual comparison of architectures. They also acknowledged that using the tool would save them time writing code to run fine tuning experiments. We describe participant feedback on individual features of the system and then outline two additional features added to address these concerns.

6.1 Participant Feedback

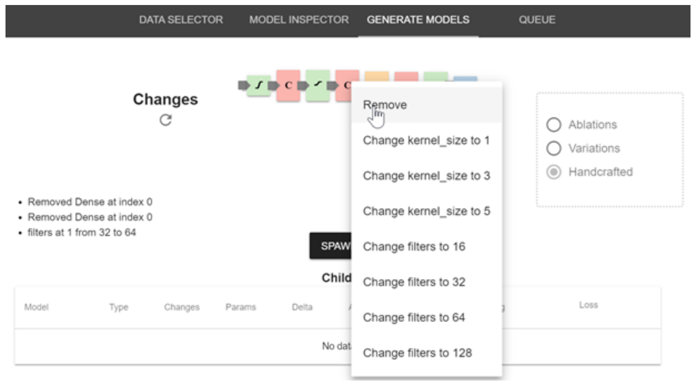
Model Overview: All participants made extensive use of the Model Overview with interpretable axes, seen in Figure 4b(A), to find baseline models. Two participants started by selecting the model with the highest accuracy irrespective of parameter size, while one participant selected smaller models first, noting that they start with smaller models when they manually select architectures: *"My intuition is to start with simple models, not try a bunch of random models, using your Model Overview."* Another participant noted that rather than start with the model with the highest accuracy, they *"thought it would make more sense to find a small model that is doing almost as well and then try to change it."*

Two participants appreciated using the Model Overview based on prediction similarity. One noted "*To me, exploring the models in that space seems like a very appealing thing to do. ... To be able to grab a subselection of them and be able to at a glance see how they are different, how do the architectures differ?*". Another participant used the model view in trying to find a small architecture for the second task that performed well on cats and birds: "*instead of looking at every model, I start with a model good at birds, then look at prediction similarity. Since it does good on birds, I'm assuming similar models do well on birds as well*". That participant explored in the neighborhood of their baseline model for a model that also performed well on cats.

Model drawer and inspection: Each participant moved multiple interesting models into the Model Drawer, and then inspected each model in the Model Inspector. They all used the confusion matrix to detect any poor qualities about models. Several participants ignored or discarded models that had all zeros in a single row which indicated that the model never predicted an instance to be that class across the entire testing dataset.

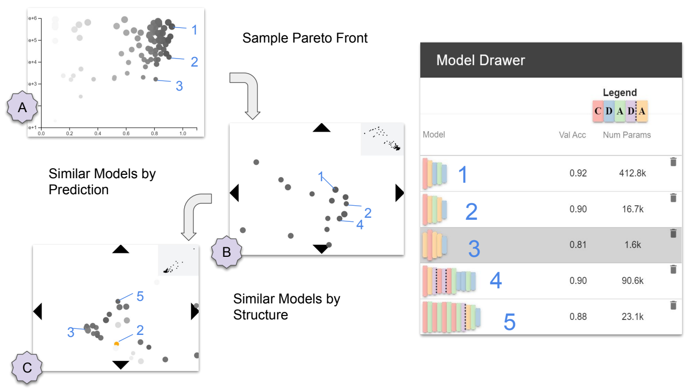
Generation of new models: While some participants found the ablation studies interesting, one participant noted that some ablations were a waste of resources: "*I basically don't want my system to waste time training models that I know will be worse... For example, removing the convolutional layer.*". Some participants used their own background and experience to inform which variations they did, while others used the Model Overview and Model Drawer to discover interesting directions to do variations in. When viewing two architectures with similar accuracy but very different sizes, a participant commented "*I can visually tell, the only difference I see is a pink color. It's a nice way to learn that dense layers add a lot of parameters.*"

All participants expressed a desire to have more control over the construction of new models. This would allow them to do more acute experimentation once they had explored in the neighborhood of an interesting baseline model. One participant described it as the need for more control over the model generation process: "*I think we need more customization on the architecture. Currently, everything is rough control ... Of course for exploring the search space, rough control would be more helpful. But for us to understand the relation [between architecture and performance], sometimes we need precise control.*" All participants noted that relying on rough control resulted in many models being spawned that were not of interest to them, especially once they had spent some time exploring the model space and knew what



- Removed Dense at index 0
- Removed Dense at index 0
- filters at 1 from 32 to 64

Fig. 5: The ability to handcraft models was added based on feedback from a validation study with model builders. Starting from a model baseline, users can remove, add, or modify any layer in the model by clicking on a layer or connections between layers. This provides fine-grained control over the models that are generated.



Model	Val Acc	Num Params
1	0.92	412.8k
2	0.90	16.7k
3	0.81	1.6k
4	0.90	90.6k
5	0.88	23.1k

Fig. 6: In our use case, the model builder first samples models 1, 2, and 3 on the pareto front of accuracy vs. model size. He then selects models 4 and 5 from the two alternative Model Overviews provided.

kind of model they wanted to generate.

6.2 System Updates

The feedback from the expert validation study led to two changes to the system. Both changes allow for more fine-grained control over which models were generated, both to allow for more precise experimentation and to reduce the number of models that need to be trained.

- **Change C1: Creating Handcrafted Models:** While variations proved useful for seeing more models in a small neighborhood in the model space, participants expressed frustration at not being able to explicitly create particular architectures. To address this, we added the handcrafted model control, seen in Figure 5. Users see the same SNAC used in the Ablations and Variations controls, but with additional handles preceding each layer. By clicking on the layer itself, users can select to either remove a layer or reparameterize it. By clicking on the handles preceding each layer, the user can choose to add a layer of any type.
- **Change C2: Subselections of ablations:** Two participants found that the ablations tool wasted time by generating models that weren't particularly of interest to the user. We added a brushing selector, seen in Figure 3a to allow the user to select *which* layers were to be used in ablations, so that the user could quickly run ablations on only a subset of the model.

7 Use Case: Classifying Sketches

To validate the new features suggested by the study, we present a use case for generating a performant, small model for an image classification dataset. In this use case, we refer to tasks T1, T2, and T3 supported by our system as outlined in section 4.

Leon is a data scientist working for a non-governmental organization that researches civil unrest around the world. He is tasked with building a mobile app for collecting and categorizing graffiti, and would like to use a neural network for classifying sketched shapes. Because his organization would like to gather data from all over the world, the application must be performant on a wide swath of mobile devices. As a result, he needs to consider the tradeoff between model size and model accuracy.

Data: He downloads a portion of the *Quick Draw* dataset to use as training data for his image classifier. Quick Draw is a collection of millions of sketches of 50 different object classes gathered by Google [1]. Rather than download the entire dataset, Leon downloads 16,000 training images and 4,000 training images from each of 10 classes that are commonly found in graffiti to serve as training data[5]. Overnight, he uses REMAP to auto-generate an initial set of 100 models, and the next day, he loads up REMAP to begin his model search.

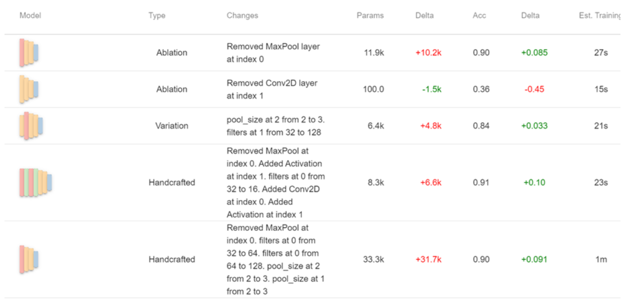
Search for baselines in the Model Overview: To find a set of baseline models [T1], he starts with the default Model Overview, seen in Figure 6A. He sees that there are many models that achieve at or above 90% accuracy, but they appear to have many parameters. He samples three models from the pareto front, two which have the high accuracy he desires and one which has an order of magnitude less parameters. He switches the model view to lay out models based on performance prediction similarity (Figure 6B) and hovers the mouse around the neighborhood of his selected models to see what alternative architectures could result in similarly good performance, and adds an additional model which has multiple convolutional and dense layers, as well as some dropout layers. Lastly, he switches the model view to lay out models based on structural similarity (Figure 6C) to see how small differences in architecture correspond to changes in either accuracy or parameters [T3]. He selects a fifth model which differs from his previously selected models in that it spreads its convolutional filters over multiple layers instead of concentrating them in a single initial layer.

Ablations: He decides to start with the smallest model, model 3, since it has reasonably high accuracy of 81% and a very small amount of parameters, approximately 1600. Having chosen a baseline, he moves on to generate local, constrained searches in the neighborhood of the baseline [T2]. After checking in the Model Inspector that the model performs reasonably well on all classes, he runs ablations on this model and sees that removing the first and last max pool layers increased both accuracy and the number of parameters. He notes that, with an accuracy of 90% and 11.9k parameters, the model resulting from removing the first max pool layer is now on the pareto front between validation accuracy and number of parameters, so he adds it to his Model Drawer for further consideration.

Variations: While the ablations indicated that he may want to remove some of the pooling layers, he wants to see the effects of various other modifications to his baseline model. He decides to generate variations of all kinds (*prepend, remove, replace, reparameterize*) along the pooling layers, and also allow for reparameterization of the convolutional layer. He generates 10 new variations from those instructions, and by looking at their results, sees that increasing the number of convolutional filters results in too many parameters, but this can be compensated for by also increasing the pool size.

Handcrafting Models: After developing an understanding of the model space, he generates some handcrafted models. He removes the first max pooling layer because that helped in the ablation studies. He then creates three new models from this template. First, he splits the starting convolutional layer into three convolutional layers with fewer filters, to be more like model 5. He then tries adding dropout, to

[5]For this use case, we used the 10 most convergent classes in Quick Draw as identified by Strobelt et al. [59]



Model	Type	Changes	Params	Delta	Acc	Delta	Est. Training
	Ablation	Removed MaxPool layer at index 0	11.9k	+10.2k	0.90	+0.085	27s
	Ablation	Removed Conv2D layer at index 1	100.0	-1.5k	0.36	-0.45	15s
	Variation	pool_size at 2 from 2 to 3. filters at 1 from 32 to 128	6.4k	+4.8k	0.84	+0.033	21s
	Handcrafted	Removed MaxPool at index 0. Added Activation at index 1. filters at 0 from 32 to 16. Added Conv2D at index 0. Added Activation at index 1	8.3k	+6.6k	0.91	+0.10	23s
	Handcrafted	Removed MaxPool at index 0. filters at 0 from 32 to 64. filters at 0 from 64 to 128. pool_size at 2 from 2 to 3. pool_size at 1 from 2 to 3	33.3k	+31.7k	0.90	+0.091	1m

Fig. 7: After generating ablations, variations, and several handcrafted models, the model builder compares all discovered models and chooses the model in the fourth row, because of its high accuracy and low number of parameters.

be more like model 4. Lastly, he creates a model with activations like model 2, and different options chosen for pooling layers and kernels inspired by the variations. The trained results can be seen in Figure 7.

Result: Leon eventually decides on using an architecture with 91% accuracy and only 8.3k parameters, seen in the fourth row of Figure 6. This model has comparable accuracy to models 1 and 2 that were initially chosen from the pareto front, seen in Figure 6, but drastically fewer parameters than model 1 (412.8k) and model 2 (16.7k). As a result, the architecture found by Leon can be deployed on older technology and classify images faster than any of the initially sampled models.

8 Discussion

8.1 Human-in-the-Loop Neural Architecture Search

Our experience and study suggested that the presence of a human-in-the-loop benefited the discovery of neural architectures. However, a common pattern in deep learning research is for applications to start with the neural network as an independent component in a set of semantic modules, only for subsequent research to point out that subsuming all components into the neural network and training it end-to-end results in superior performance. As an example, the *R-CNN* method for object recognition dramatically outperformed baselines for object detection using a CNN in concert with a softmax classifier and multiple bounding box regressors [14]; however, its performance was eclipsed only one year later by *Fast R-CNN*, which absorbed the classifiers and regressors into the neural network [13, 71]. This suggests that the user processes in REMAP , such as selecting models on the pareto front and running certain ablations and variations, could be automated, and the whole process run end to end as a single optimization without a human-in-the-loop. Ultimately, this perspective ignores the tradeoffs that users are able to make; users can very quickly and efficiently narrow the search space to only a small subset of interesting baselines based on a number of criteria that are not available to the automated methods. These include fuzzy constraints on the number of parameters, a fuzzy cost function that differs per class and instance, and domain knowledge of the deployment scenario of the model. For this reason, we advocate that the human has a valuable role when searching for a neural architecture using REMAP .

8.2 Generalizability

The workflow of REMAP is generalizable to other types of automated machine learning and model searches beyond neural networks. The two primary components of REMAP are a set of projections of models and a local sampling method to generate models in a neighborhood of a baseline model. As long as these two components can be defined for a model space, the workflow of REMAP is applicable. Of the three projections used, both the semantically meaningful projection of accuracy vs. number of parameters and the prediction similarity distance metrics are generalizable to any machine learning model, while structural similarity distances can be easily chosen, such as the Euclidean distance between weights for a support vector machine. Similarly, random sampling in the neighborhood of a model can be done in any number of ways; if the model space is differentiable, gradient-based techniques can be used to sample in the direction of accurate or small models.

8.3 Scalability

In order to facilitate human-in-the-loop-neural architecture search, REMAP must make several constraints on its model space. It limits the size of the architectures it discovers so that they can be trained in a reasonable amount of time while the user is engaged with the application. In certain domains, however, the tradeoff between accuracy and size of the model is very different; stakeholders don't want to sacrifice any accuracy. In that case, the cap on model size in REMAP could be removed, and REMAP could be used to find large networks that take many hours to train. It isn't feasible to expect a user to stay *in situ* the entire time while REMAP trained the several dozen models needed to enable architecture discovery. Instead, a dashboard-like experience, easily viewable in a casual setting on a small screen such as a phone might be preferable. In general, the types of user experiences used in visual analytics tools for machine learning models may have to be adapted to the scale of time necessary for constructing and searching through industry-level neural networks.

The visual encoding used for neural network architectures, SNACs, can only display network architectures that are linked lists, which leaves out some newer types of architectures that have *skip connections*, which are additional linkages between layers. This problem could be solved by improving the encoding to communicate skip connections. Ultimately, supporting every possible network architecture amounts to supporting arbitrary graphs, and there is no space-efficient way to do so without losing information. For that reason, we limit the scope in this project to network architectures that are linked lists, because they are simpler to understand and are a common architecture that are more performant than non-neural network models for image classification problems.

9 Conclusion

Neural networks can be difficult to use because choosing an architecture requires tedious and time consuming manual experimentation. Alternatively, automated algorithms for neural architecture search can be used, but they require large computational resources and cannot accommodate soft constraints such as trading off accuracy for model size or trading off on performance between classes. We present REMAP, a visual analytics tool that allows a model builder to discover a deep learning model quickly via exploration and rapid experimentation of neural network architectures and their parameters. REMAP enables users to quickly search for baseline models through a visual overview, visually compare subsets of those models to understand small, local differences in architectures, and then generate local, constrained searches to fine tune architectures. Through a design study with four model builders, we derive a set of design goals. We provide a use case in building a small image classifier for identifying sketches in graffiti that is small enough to used on even very old mobile devices. We demonstrate that the semi-automated approach of REMAP allows users to discover architectures quicker and easier than through manual experimentation or fully automated search.

Acknowledgments

We would like to thank Subhajit Das, who provided code for confusion matrices, and Kirthevasan Kandasamy, who gave assistance incorporating the OTMANN distance metric. Support for the research is partially provided by DARPA FA8750-17-2-0107 and NSF CAREER IIS-1452977. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

References

[1] Quick, Draw! can a neural network learn to recognize doodling? `https://quickdraw.withgoogle.com/`. Accessed: 2019-03-30.

[2] J. Adebayo, J. Gilmer, M. Muelly, I. J. Goodfellow, M. Hardt, and B. Kim. Sanity checks for saliency maps. *CoRR*, abs/1810.03292, 2018.

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

[4] B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. *CoRR*, abs/1611.02167, 2016.

[5] J. Bergstra, D. Yamins, and D. D. Cox. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in Science Conference*, pp. 13–20, 2013.

[6] 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.

[7] D. Cashman, G. Patterson, A. Mosca, N. Watts, S. Robinson, and R. Chang. Rnnbow: Visualizing learning via backpropagation gradients in rnns. *IEEE Computer Graphics and Applications*, 38(6):39–50, 2018.

[8] M. Cavallo and . Demiralp. Clustrophile 2: Guided visual clustering analysis. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):267–276, 2019.

[9] J. Choo, H. Lee, J. Kihm, and H. Park. ivisclassifier: An interactive visual analytics system for classification based on supervised dimension reduction. In *Visual Analytics Science and Technology (VAST), 2010 IEEE Symposium on*, pp. 27–34. IEEE, 2010.

[10] 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.

[11] S. Das, D. Cashman, R. Chang, and A. Endert. Beames: Interactive multi-model steering, selection, and inspection for regression tasks. *Symposium on Visualization Data Science*, 2018.

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

[13] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.

[14] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

[15] M. Gleicher. Considerations for visualizing comparison. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):413–423, 2018.

[16] I. Guyon, K. Bennett, G. Cawley, H. J. Escalante, S. Escalera, T. K. Ho, N. Macia, B. Ray, M. Saeed, A. Statnikov, et al. Design of the 2015 chalearn automl challenge. In *Neural Networks (IJCNN), 2015 International Joint Conference on*, pp. 1–8. IEEE, 2015.

[17] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[18] F. M. Hohman, M. Kahng, R. Pienta, and D. H. Chau. Visual analytics in deep learning: An interrogative survey for the next frontiers. *IEEE Transactions on Visualization and Computer Graphics*, 2018.

[19] D. H. Jeong, C. Ziemkiewicz, B. Fisher, W. Ribarsky, and R. Chang. ipca: An interactive system for pca-based visual analytics. In *Computer Graphics Forum*, vol. 28, pp. 767–774. Wiley Online Library, 2009.

[20] H. Jin, Q. Song, and X. Hu. Auto-keras: Efficient neural architecture search with network morphism. *arXiv preprint arXiv:1806.10282*, 2018.

[21] M. Kahng, P. Y. Andrews, A. Kalro, and D. H. P. Chau. Activis: Visual exploration of industry-scale deep neural network models. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):88–97, 2018.

[22] M. Kahng, D. Fang, and D. H. P. Chau. Visual exploration of machine learning results using data cube analysis. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, HILDA '16, pp. 1:1–1:6. ACM, New York, NY, USA, 2016. doi: 10.1145/2939502.2939503

[23] M. Kahng, N. Thorat, D. H. P. Chau, F. B. Viégas, and M. Wattenberg. GAN lab: Understanding complex deep generative models using interactive visual experimentation. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):310–320, 2019.

[24] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing. Neural architecture search with bayesian optimisation and optimal transport. In *Advances in Neural Information Processing Systems*, pp. 2020–2029, 2018.

[25] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1725–1732, 2014.

[26] B. Komer, J. Bergstra, and C. Eliasmith. Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn. In *ICML workshop on AutoML*, 2014.

[27] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *Journal of Machine Learning Research*, 17:1–5, 2016.

[28] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

[30] B. C. Kwon, B. Eysenbach, J. Verma, K. Ng, C. De Filippi, W. F. Stewart, and A. Perer. Clustervision: Visual supervision of unsupervised clustering. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):142–151, 2018.

[31] S. Lai, L. Xu, K. Liu, and J. Zhao. Recurrent convolutional neural networks for text classification. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[32] Y. LeCun, Y. Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[33] F.-F. Li and J. Li. Cloud AutoML: Making AI accessible to every business. https://www.blog.google/topics/google-cloud/cloud-automl-making-ai-accessible-every-business/. Accessed: 2018-03-29.

[34] Z. C. Lipton. The mythos of model interpretability. *arXiv preprint arXiv:1606.03490*, 2016.

[35] H. Liu, K. Simonyan, and Y. Yang. DARTS: differentiable architecture search. *CoRR*, abs/1806.09055, 2018.

[36] M. Liu, J. Shi, K. Cao, J. Zhu, and S. Liu. Analyzing the training processes of deep generative models. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):77–87, 2018.

[37] M. Liu, J. Shi, Z. Li, C. Li, J. Zhu, and S. Liu. Towards better analysis of deep convolutional neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):91–100, 2017.

[38] 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.

[39] J. Lu, J. Yang, D. Batra, and D. Parikh. Hierarchical question-image co-attention for visual question answering. In *Advances In Neural Information Processing Systems*, pp. 289–297, 2016.

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

[41] 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, 2018.

[42] 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.

[43] 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, 2007.

[44] E. J. 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, 2013.

[45] C. Olah, A. Satyanarayan, I. Johnson, S. Carter, L. Schubert, K. Ye, and A. Mordvintsev. The building blocks of interpretability. *Distill*, 2018. https://distill.pub/2018/building-blocks. doi: 10.23915/distill.00010

[46] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[47] N. Pezzotti, T. Höllt, J. Van Gemert, B. P. Lelieveldt, E. Eisemann, and A. Vilanova. Deepeyes: Progressive visual analytics for designing deep neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):98–108, 2017.

[48] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. *CoRR*, abs/1802.01548, 2018.

[49] 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, 2018.

[50] 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.

[51] M. Sedlmair, C. Heinzl, S. Bruckner, H. Piringer, and T. Mller. Visual parameter space analysis: A conceptual framework. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2161–2170, Dec 2014. doi: 10.1109/TVCG.2014.2346321

[52] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *CoRR*, abs/1610.02391, 2016.

[53] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034, 2013.

[54] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[55] D. Smilkov and S. Carter. Tensorflow Playground. `https://playground.tensorflow.org/`. Accessed: 2019-03-31.

[56] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pp. 2951–2959, 2012.

[57] H. Strobelt, S. Gehrmann, M. Behrisch, A. Perer, H. Pfister, and A. 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.

[58] H. Strobelt, S. Gehrmann, H. Pfister, and A. M. Rush. Lstmvis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):667–676, 2018.

[59] H. Strobelt, E. Phibbs, and M. Martino. Ffqd-mnist – Forma Fluens quickdraw model decay indicator dataset. `http://www.formafluens.io/client/mix.html`. Version: 0.1.

[60] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, et al. Going deeper with convolutions. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2015.

[61] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 847–855. ACM, 2013.

[62] M. W. Trosset and C. E. Priebe. The out-of-sample problem for classical multidimensional scaling. *Computational statistics & data analysis*, 52(10):4635–4642, 2008.

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

[64] S. Van Den Elzen and J. J. van Wijk. Baobabview: Interactive construction and analysis of decision trees. In *Visual Analytics Science and Technology (VAST), 2011 IEEE Conference on*, pp. 151–160. IEEE, 2011.

[65] J. Wang, L. Gou, H.-W. Shen, and H. Yang. Dqnviz: A visual analytics approach to understand deep q-networks. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):288–298, 2019.

[66] K. Wongsuphasawat, D. Smilkov, J. Wexler, J. Wilson, D. Mané, D. Fritz, D. Krishnan, F. B. Viégas, and M. Wattenberg. Visualizing dataflow graphs of deep learning models in tensorflow. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):1–12, 2018.

[67] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pp. 2048–2057, 2015.

[68] J. Yosinski, J. Clune, A. M. Nguyen, T. J. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. *CoRR*, abs/1506.06579, 2015.

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

[70] 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*, 2018.

[71] 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*, 2019.

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

[73] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2017.