Supplementary Material for Teacher Guided Architecture Search

Pouya Bashivan
Department of Brain and Cognitive Sciences
McGovern Institute for Brain Research
MIT
bashivan@mit.edu

Mark Tensen
University of Amsterdam
mark.tensen@student.uva.nl

James J DiCarlo
Department of Brain and Cognitive Sciences and
McGovern Institute for Brain Research
MIT
dicarlo@mit.edu

Hyperparameter Search with Reinforcement Learning (RL)

We follow the method proposed by [14] to learn the probability of hyperparameter choices ($\mathcal{X} = x_1, x_2, ..., x_n$) that maximize the unknown but observable reward function $f : \mathcal{X} \rightarrow \mathbb{R}$. A 2-layer long short-term memory (LSTM) is used as the controller that chooses each hyperparameter in the network at every unrolling step. The LSTM network, models the conditional probability distribution of optimal hyperparameter choices as a function of all previous choices $P(x_j|x_1, x_2, ..., x_{j-1}, \theta)$ in which θ is the set of all tunable parameters in the LSTM network. Since a differentiable loss function is not known for this problem, usual maximum likelihood methods could not be used in this setting. Instead parameters are optimized through reinforcement learning based approaches (e.g. REINFORCE [12]) by increasing the likelihood of each hyperparameter choice according to the reward (score) computed for each sampled network (or a batch of sampled networks). Relative to [14], we made two modifications. First, since the order of dependencies between different hyperparameters in each layer/block is arbitrary, we ran the LSTM controller for one step per layer (instead of once per hyper-parameter). This results in shorter choice sequences generated by the LSTM controller and therefore shorter sequence dependencies. Second, we chose a Boltzman policy method for action selection to allow the search to continue the exploration throughout the search experiment. Hyperparameter values were selected according to the probability distribution over all action choices. Compared to ϵ-greedy method, following the softmax policy reduces the likelihood of sub-optimal actions throughout the training.

For each hyperparameter, choice probability is computed using a linear transformation (e.g. $W_{K_h}, W_{N_{filters}}$) from LSTM output at the last layer (h_l^2) followed by a softmax. To reduce the number of tunable parameters and more generalization across layers, we used shared parameters between layers.

$$\begin{aligned} &\hat{P}_{l,x} = \text{softmax}(W_t^T h_l^2) \\ &l \in \{1, 2, ..., N_l\} \\ &t \in \{K_h, K_w, N_{filters}, \text{stride, normalization, activation}\} \end{aligned} \tag{1}$$

Probability distribution over possible number of layers is formulated as a function of the first output value of the LSTM ($\hat{P}_{N_l} = \text{softmax}(W_{N_l}^T h_0^2)$). In addition to layers' hyperparameters we also search over layers' connections. Similar to the approach taken in [14] we formulated the probability of a connection between layer *i* and *j* as a function of the state of the LSTM at each of these layers (h_i^2, h_j^2).

$$\hat{P}_{i,j}^c = \text{sigmoid}(W_{src}^T h_i^2 + W_{dst}^T h_j^2) \tag{2}$$

where $\hat{P}_{i,j}^c$ represents the probability of a connection between layer *i* output to *j*'s input. W_{src} and W_{dst} are tunable parameters that link the hidden state of LSTM to probability of a connection existing between the two layers.

Hyperparameter Search with Tree of Parzen Estimators (TPE)

Sequential Model-Based Optimization (SMBO) [7] approaches are numerical methods used to optimize a given score function $f : \mathcal{X} \rightarrow \mathbb{R}$. They are usually applied in settings where evaluating the function at each point is costly

and it's important to minimize the number of evaluations to reach the optimal value. Various SMBO approaches were previously proposed [3, 1] and some have been used for hyperparameter optimization in neural networks [2, 4, 9]. Bayesian SMBO approaches model the posterior or conditional probability distribution of values (scores) and use a criteria to iteratively suggest new samples while the probability distribution is updated to incorporate the history of previous sample tuples (x, y) where $x = (x^{(1)}, ..., x^{(n)})$ is a sample hyperparameter vector and y is the received score (or loss). Here we adopted Tree of Parzen Estimators (TPE) because of its intuitiveness and successful application in various domains with high dimensional spaces. Unlike most other Bayesian SMBO methods that directly model the posterior distribution of values $P(y|x)$, TPE models the conditional distribution $P(x|y)$ with two non-parametric densities.

$$P(x|y) = \begin{cases} l(x) & \forall\, y \leq y^* \\ g(x) & \forall\, y > y^* \end{cases} \tag{3}$$

We consider y to be the loss value which we are trying to minimize (e.g. error rate of a network on a given task). For simplicity, value of y^* could be taken as some quantile of values observed so far (γ). At every iteration, TPE fits a kernel density estimator with Gaussian kernels to subset of observed samples with lowest loss value ($l(x)$) and another to those with highest loss ($g(x)$). Ideally we want to find x that minimizes y. Expected Improvement (EI) is the expected reduction in $f(x)$ compared to threshold y^* under current model of f. Maximizing EI, encourages the model to further explore parts of the space which lead to lower loss values and can be used to suggest new hyperparameter samples.

$$\begin{aligned} EI(x) &= \int_{-\infty}^{y^*} (y^* - y) P(y|x) dy \\ &= \frac{\int_{-\infty}^{y^*} (y^* - y) P(x|y) P(y) dy}{P(x)} \end{aligned} \tag{4}$$

Given that $P(y < y^*) = \gamma$ and $P(x|y) = l(x)$ for $y < y^*$, it has been shown [2] that EI would be proportional to $\left(\gamma + \frac{g(x)}{l(x)}(1-\gamma)\right)^{-1}$. Therefore the EI criterion can be maximized by taking samples with minimum probability under $g(x)$ and maximum probability under $l(x)$. For simplicity, at every iteration n_d samples are drawn from $l(x)$ and the hyperparameter choice with lowest $g(x)/l(x)$ ratio is suggested as the next sample.

Alternative Teacher Network - NASNet

We examined the effect of choosing an alternative teacher network, namely NASNet and performed a set of analyses similar to those done on ResNet. We observed that similar to ResNet, early layers are better predictors of the mature performance during early stages of the training. With additional training, the premature performance becomes a better single-predictor of the mature performance but during most of the training the combined P+TG score best predicts the mature performance (Figure 1-left). We also varied the "TG" weight factor and found that compared to ResNet, higher α values led to increased gains in predicting the mature performance. $\alpha = 5$ was used to compute the P+TG scores shown in Figure 1.

Overall, we found that NASNet representations were significantly better predictors of mature performance for all evaluated time points during training when compared to ResNet (Figure 1-right).

Datasets and Preprocessing

CIFAR: We followed the standard image preprocessing for CIFAR labeled dataset, a 100-way object classification task [6]. Images were zero-padded to size 40×40. A random crop of size 32×32 was selected, randomly flipped along the vertical axis, and standardized over all pixel values in each image to have zero mean and standard deviation of 1. We split the training set into training set (45,000 images) and a validation set (5,000 images) by random selection.

Imagenet: We used standard VGG preprocessing [11] on images from Imagenet training set. During training, images were resized to have their smaller side match a random number between 256 and 512 while preserving the aspect ratio. A random crop of size 224 was then cut out from the image and randomly flipped along the central vertical axis. The central crop of size 224 was used for evaluation.

Details of Search Algorithms

RL Search Algorithm: We used a two-layer LSTM with 32 hidden units in each layer as the controller. Parameters were trained using Adam optimizer [8] with a batch size of 5. For all searches, the learning rate was 0.001, and the Adam first momentum coefficient was zero $\beta_1 = 0$. Gradients were clipped according to global gradient norm with a clipping value of 1 [10].

TPE Search Algorithm: We used the python implementation of TPE hyperparameter search from HyperOpt package [5]. We employed the linear sample forgetting as suggested in [4] and set the threshold $y^* = \sqrt{N}/4$ for the set of N observed samples. Each search run started with 20 random samples and continued with TPE suggestion algorithm. At every iteration, $n_d = 24$ draws were taken from $l(x)$ and choice of hyperparameter $argmin_i g(x_i)/l(x_i)$ was used as the next sample (see section 3.3 in the main text).

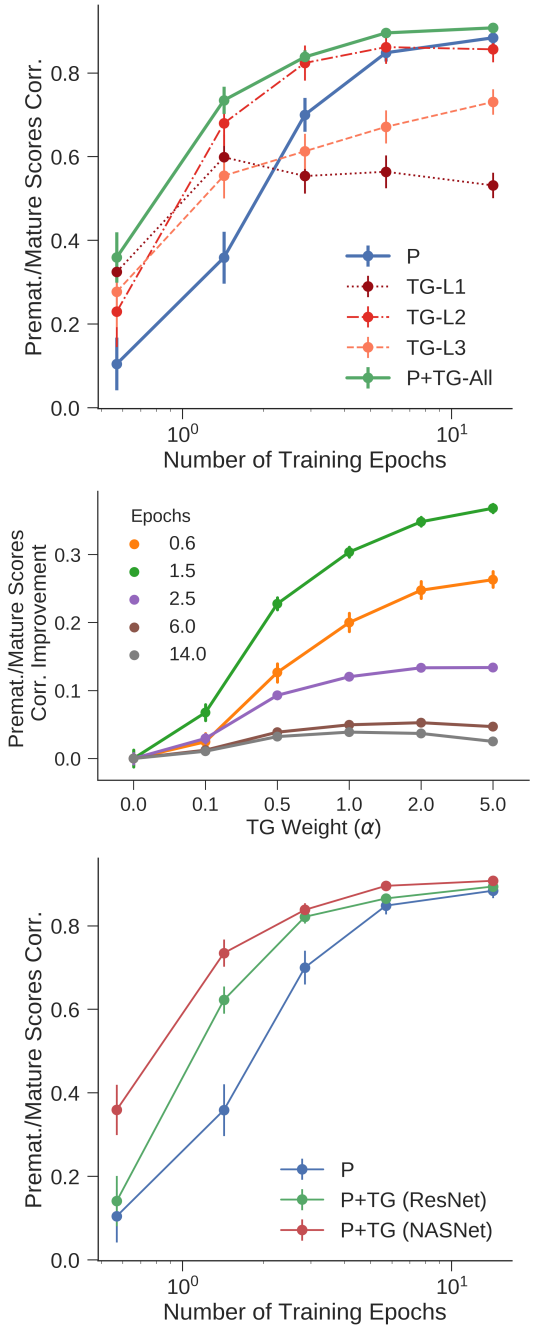

Figure 1. (top) Comparison of single layer and combined RDMs with premature performance as predictors of mature performance on NASNet. P+TG was computed using $\alpha = 5$. (middle) Gain in predicting the mature performance with varying TG weight. (bottom) Comparison of combined RDM scores using two alternative teacher models at various stages of training. α values of 1 and 5 were used for ResNet and NASNet respectively.

Experimental Details for Search in the Space of Convolutional Networks

Search Space: Similar to [14] we defined the hyperparameter space as the following independent choices for each layer: $N_{filters} \in [32, 64, 128]$, $(K_{width}, K_{height}) \in [1, 3, 5, 7]$, $K_{stride} \in [1, 2]$, activation $\in [Identity, ReLU]$, normalization $\in [none, BN]$. In addition we searched over number of layers ($N_{layers} \in [1, N_L]$) and possible connections between the layers. In this space of CNNs, the input to every layer could have originated from the input image or the output of any of the previous layers. We considered two particular spaces in our experiments that differed in the value of N_L (=10 or 20).

CIFAR Training: Selected networks were trained on CIFAR training set (45k samples) from random initial weights using SGD with Nesterov momentum of 0.9 for 300 epochs on the training set. The initial learning rate was 0.1 and was divided by 10 after every 100 epochs. Mature performance was then evaluated on the validation set (above).

Experimental Details for Search in the Space of Convolutional Cells

Search Space: We used the same search space and network generation procedure as in [15, 9] with the exception that we added two extra hyperparameters which could force each of the cell inputs (from previous cell or the one prior to that) to be directly concatenated in the output of the cell even if they were already connected to some of the blocks in the cell. This extra hyperparameter choice was motivated by the open-source implementation of NASNet at the time of conducting the search experiments that contained similar connections[1].

Each cell receives two inputs which are the outputs of the previous two cells. In early layers, the missing inputs are substituted by the input image. Each cell consists of B blocks with a prespecified structure. Each block receives two inputs, an operation is applied on each input independently and the results are added together to form the output of the block. The search algorithm picks each of the operations and inputs for every block in the cell. Operations are selected from a pool of 8 possible choices: {identity, 3×3 average pooling, 3×3 max pooling, 3×3 dilated convolution, 1×7 followed by 7×1 convolution, 3×3 depthwise-separable convolution, 5×5 depthwise-separable convolution, 7×7 depthwise-separable convolution}.

Imagenet Training: For our Imagenet training experiments, we used a batch size of 128 images of size 224×224 pixels. Each batch was divided between two GPUs and the gradients computed on each half were averaged before updating the weights. We used an initial learning rate of 0.1 with a decay of 0.1 after every 15 epochs. Each network was trained for 40 epochs on the Imagenet training set and

[1] available at https://github.com/tensorflow/models/blob/376dc8dd0999e6333514bcb8a6beef2b5b1bb8da/research/slim/nets/nasnet/nasnet_utils.py

validated on the central crop for all images from Imagenet validation. No dropout or drop-path was used when training the networks. RMSProp optimizer with a decay rate of 0.9 and momentum rate of 0.9 was used during training and gradients were normalized by their global norm when the norm value exceeded a threshold of 10. L2-norm regularizer was applied on all trainable weights with a weight decay rate of 4×10^{-5}.

CIFAR Training: The networks were trained on CIFAR10/CIFAR100 training set including all 50,000 samples for 600 epochs with an initial learning rate of 0.025 and a single period cosine decay [15]. We used SGD with Nesterov momentum rate of 0.9. We used L2 weight decay on all trainable weights with a rate of 5×10^{-4}. Gradient clipping similar to that used for Imagenet and a threshold of 5 was used.

Best Discovered Convolutional Cell: Figure 2 shows the structure of the best discovered cell by TG-SAGE on CIFAR100. Only four (out of ten) operations contain trainable weights and there are several bypass connections in the cell.

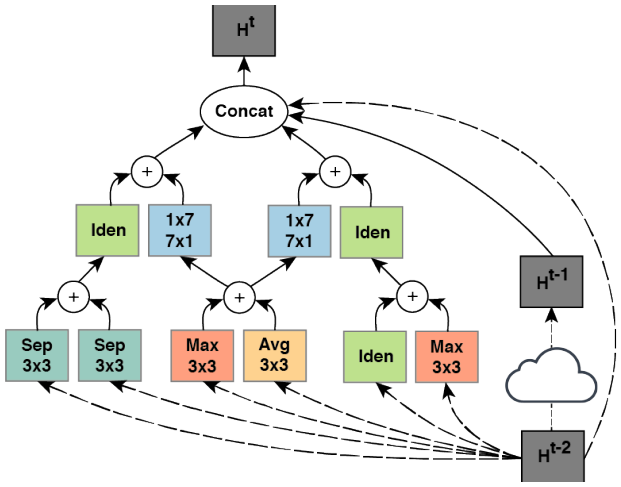

Figure 2. SAGENet - Structure of the best cell discovered during the search with TG-SAGE.

Neural Measurements from Macaque Monkeys

We used a dataset of neural spiking activity for a population of 296 neural sites in two awake behaving macaque monkeys in response to 5760 images [13]. Neural data were collected using parallel microelectrode arrays that were chronically implanted on the cortical surface in area V4 and IT. Fixating animals were presented with images for 100ms, and the neural response patterns were obtained by averaging the spike counts in the time window of 70-170ms post stimulus onset. To enhance the signal-to-noise ratio, each image was presented to each monkey between 21-50 times and the average response pattern across all presentation were considered for each image. The 296 recorded sites were partitioned into three cortical regions (V4, posterior-IT, and anterior-IT) and a RDM was calculated for each region.

The image set consisted of a total of 5760 images. Each image contained a 3D rendered object placed on an uncorrelated natural background. The rendered objects were selected from a battery of 64 objects from 8 categories (animals, boats, cars, chairs, faces, fruits, planes, and tables) with 8 objects per category. The images were generated to include large variations in position, size, and pose of the objects and were shown within the central 8° of monkeys' visual field. Some example images are illustrated in Figure-3.

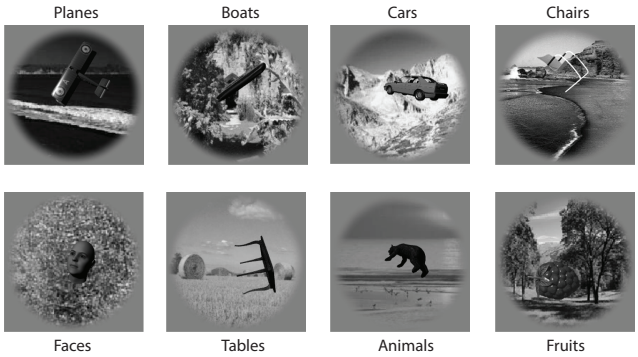

Figure 3. Example images from each of the eight object categories that were used to record neural responses.

Implementation Details

Because of heavy computational load associated with training neural networks and in particular in large-scale model training, we needed a scalable and efficient framework to facilitate the search procedure. We implemented our proposed framework in four main modules: (i) explorer, (ii) trainer, (iii) evaluator, and (iv) tracker. The explorer module contained the search algorithm. The trainer module optimized the parameters of the proposed architecture on an object recognition task using a large-scale image dataset. Once the training job was completed, the evaluator module extracted the network activations in response to a set of predetermined image-set and assessed the similarity of representations to the teacher benchmarks. The tracker module consisted of a database which tracked the details and status of every proposed architectures and acted as a bridge between all three modules.

During the search experiments, the explorer module proposes new candidate architectures and records the details in the database (tracker module). It also continuously monitors the database for newly evaluated networks. Upon receiving adequate number of samples (i.e. when a new

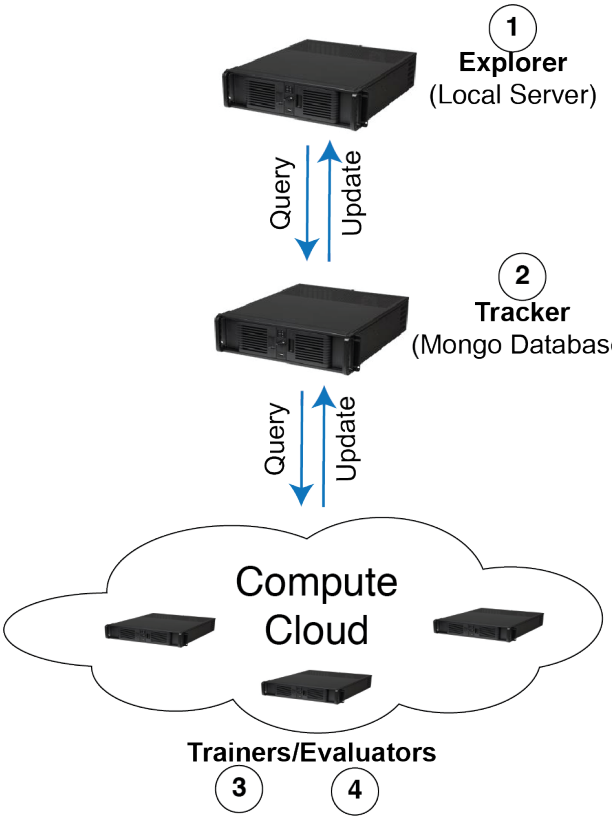

Figure 4. Implementation of a distributed framework for conducting architecture search.

batch is complete), it updates its parameters. Active workers periodically monitor the database for newly added untrained models, and train the architecture on the prespecified dataset. After the training phase is completed, the evaluator module extracts the features from all layers in response to the validation set and computes the premature-performance and RDM consistencies and writes back the results in the database. The trainer and evaluator modules are then freed up to process new candidate networks. This framework enabled us to run many worker programs on several clusters speeding up the search procedure. An overview of the implemented framework is illustrated in Figure 4. Experiments reported in this paper were run on three server clusters with up to 40 GPUs in total.

References

[1] R. Bardenet and B. Kegl. Surrogating the surrogate: accelerating gaussian-process-based global optimization with a mixture cross-entropy algorithm. In *27th International Conference on Machine Learning (ICML 2010)*, pages 55–62. Omnipress, 2010. 2

[2] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kegl. Algorithms for Hyper-Parameter Optimization. pages 1–9, 2011. 2

[3] J. Bergstra, N. Pinto, and D. Cox. Machine learning for predictive auto-tuning with boosted regression trees. In *Innovative Parallel Computing (InPar), 2012*, pages 1–9. IEEE, 2012. 2

[4] J. Bergstra, D. Yamins, and D. D. Cox. Making a Science of Model Search. pages 1–11, 2012. 2

[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*, pages 13–20. Citeseer, 2013. 2

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *Arxiv.Org*, 7(3):171–180, 2015. 2

[7] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer, 2011. 1

[8] D. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations*, pages 1–13, 2014. 2

[9] C. Liu, B. Zoph, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive Neural Architecture Search. 2017. 2, 3

[10] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training Recurrent Neural Networks. (2), 2012. 2

[11] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. pages 1–10, 2014. 2

[12] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992. 1

[13] D. L. K. Yamins, H. Hong, C. F. Cadieu, E. A. Solomon, D. Seibert, and J. J. DiCarlo. Performance-optimized hierarchical models predict neural responses in higher visual cortex. *Proceedings of the National Academy of Sciences*, 111(23):8619–8624, 2014. 4

[14] B. Zoph and Q. V. Le. Neural architecture Search With reinforcement learning. *ICLR*, 2017. 1, 3

[15] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning Transferable Architectures for Scalable Image Recognition. 10, 2017. 3, 4