QENAS: Q-Learning for Efficient Neural Architecture Search
CSC498H5F: Introduction to Reinforcement Learning (Topics)

Robert Wu RUPERT@CS.TORONTO.EDU
Department of Computer Science
University of Toronto

Rohan Jain ROHAN.JAIN@MAIL.UTORONTO.CA
Department of Mathematics
University of Toronto

1. Introduction

The problem of finding optimal deep learning architectures has recently been automated by neural architecture search (NAS) algorithms. These algorithms continually sample operations from a predefined search space to construct neural networks to optimize a performance metric over time, eventually converging to better child architectures. This intuitive idea greatly reduces human intervention by restricting human bias in architecture engineering to just the selection of the predefined search space (Elsken et al., 2019). Although NAS has the potential to revolutionize architecture search across many applications, human selection of the search space remains a security risk that needs to be evaluated before NAS can be deployed in security-critical domains. While NAS has been studied to further develop more adversarially robust networks through addition of dense connections (Guo et al., 2020), little work has been done in the past to assess the adversarial robustness of NAS itself. In this project, we are focused on a particular REINFORCE based NAS algorithm known as Efficient NAS (ENAS) (Pham et al., 2018), which significantly cuts down on the expensive computational complexity of NAS with reinforcement learning (RL) (Zoph and Le, 2016) by incorporating weight sharing.

Search phase analysis has shown that computationally efficient algorithms such as ENAS are worse at truly ranking child networks due to their reliance on weight sharing (Yu et al., 2019). In Wu et al. (2021), we validated these concerns by evaluating the robustness of ENAS against data-agnostic search space poisoning (SSP) attacks on the CIFAR-10 dataset. The domain of ENAS is a *macro search space* of network operations that form convolutional neural networks (CNNs), whose construction is determined by how the REINFORCE-based controller chooses network operations and skip connections. In (Pham et al., 2018), they define the pre-optimized ENAS search space $\hat{\mathcal{E}}$ = {Identity, 3x3 Separable Convolution, 5x5 Separable Convolution, Max Pooling (3x3), Average Pooling (3x3)}. In learning tasks such as CIFAR-10/100 classification, the resulting networks vary in performance, which the controller treats as a surrogate reward. The quality of the controller's decision-making is affected by not only the system's hyperparameters, but also the search space itself. We hypothesize that there are many ways ENAS (and NAS algorithms in general) can still be improved to produce better networks and be more robust or more efficient.

2. Related Works

A comprehensive overview of NAS algorithms can be found in Wistuba et al. (2019) and Elsken et al. (2019), with Chakraborty et al. (2018) summarising advances in adversarial machine learning including poisoning attacks. NAS algorithms have recently been employed in healthcare and applied in various clinical settings for diseases like COVID-19, cancer and cystic fibrosis (van der Schaar, 2020). Furthermore, architectures derived from NAS procedures have shown state of the art performance, often outperforming manually created networks in semantic segmentation (Chen et al., 2018), image classification (Real et al., 2019; Zoph et al., 2018) and object detection (Zoph et al., 2018). In the context of adversarial attacks, most traditional poisoning attacks involve injecting mislabeled examples in the training data, which is fairly limited. The expected result is higher prediction error and in some cases a complete reversal of what the network should be predicting. Some examples of traditional poisoning attacks have been executed against feature selection methods (Xiao et al., 2015), support vector machines (Biggio et al., 2012) and neural networks (Yang et al., 2017). Recent works have been improving ENAS from a pure RL perspective such as reward modifications and policy optimization. For instance, in I-ENAS (Nagy and Ábel Boros, 2021) they change the RL training process by modifying the reward of each tested architecture according to the results based on previous architectures. On CIFAR-10, I-ENAS reached higher maximum accuracy than ENAS and consistently produced better architectures, with an accuracy of 95.3%. Policy gradient methods have have made several breakthroughs in Deep RL. It's been studied that typical policy gradient methods have poor sampling efficiency and tend to have larger variance compared to other Q-Learning methods. REINFORCE is the most elementary policy gradient method, used to train the ENAS controller. In (Gallo et al., 2020), they modify the training of the ENAS controller to use Proximal Policy Optimization (PPO) instead of REINFORCE in effort to improve the sampling efficiency of ENAS. The main difference between traditional REINFORCE & PPO is that in PPO they compute the loss and update the policy's parameters achieving slightly better performance than a controller trained with REINFORCE using fewer sample architectures.

3. Preliminaries

3.1 Search Space Poisoning (SSP)

Briefly summarized, we developed SSP as a data agnostic poisoning attack on the original search space ENAS search space $\mathcal{E}$. The idea is to inject precisely designed multiset $\mathcal{P}$ of ineffective operations into $\mathcal{E}$, where we define a poisonous search space as $\hat{\mathcal{E}} := \mathcal{E} \cup \mathcal{P}$. In Wu et al. (2021), we exploited the core functionality of the ENAS controller to sample child networks from a large computational graph of operations by introducing highly ineffective local operations into the search space. We first propose multiple-instance poisoning which increases the likelihood of sampling bad operations by including duplicates of these bad operations in the search spaces. Through experimental results we discovered that biasing the search space this way resulted in final networks that are mostly comprised of these poor operations with error rates exceeding 80%. On the attacker's behalf, this requires no *a priori* knowledge of the problem domain or dataset being used, making this new approach more favourable than traditional data poisoning attacks. However, we were quick

to realize for multiple-instance poisoning attacks to perform well it requires overwhelming the original search space with up to 300 bad operations which results in a ratio of 50:1 of bad operations per each good operation which is unreasonable. The motivation was then shifted to reduce the ratio of bad to good operations down to 1:1 or even lower. In (Saxena et al., 2022), one-shot poisoning attacks were introduced where we reduced the ratio of bad to good operations to 1:3. We injected just two operations into $\hat{\mathcal{E}}$, (i) Dropout($p = 1$) & (ii) Stretched Convolution ($k = 3$, padding, dilation $= 50$), and achieved error rates shooting up to 90%. In section 4, we hypothesize a Q-Learning like based approach to defend against both multiple-instance & one-shot poisoning attacks.

3.2 REINFORCE in Neural Architecture Search

The ENAS controller is an LSTM cell which is trained using REINFORCE.

Algorithm 1 Training process for REINFORCE in ENAS

```
procedure REINFORCE
    C ← controller
    for each controller epoch:
        SAMPLE M architectures in each epoch, with a fixed policy.
        for each child model m ∈ M:
            TRAIN m
            reward ← calculate the val. accuracy
            baseline ← calculate mean val. accuracy for previous 5 controller epochs
            p ← sampled log probabilities of the controller's decisions
            h ← sampled entropy of the controller's decisions
            l ← p × (reward - baseline) + h
            Perform gradient ascent on l and update C parameters, ω.
        end
    end
```

4. Methods

The LSTM-based controller is powerful, but it's quite expensive. As we found in Wu et al. (2021), depending on the hardware, the shared CNN training step was able to consistently perform anywhere between 3 to 6 iterations per second, resulting in training times ranging between 20 to 36 hours for 300 epochs. We hypothesize that using some form of Q-Learning would be more efficient and potentially improve performance.

To formulate the Q-Learning problem, we must discuss the state space $\mathcal{S}$. In traditional Q-Learning, the state space can be conveniently described as naturally discrete or discretized. Unfortunately, after removing part or all of the LSTM, the only notion of state remaining is the parameters ω of the shared CNN. This immediately becomes complicated because ω is complex and not differentiable in the context of network construction. A potential work-around is stateless Q-Learning, which black-boxes the states. Therefore, we will will disregard state in developing the Q-table and we reference Q-values as $Q(a) := Q(\cdot, a)$. As we use Q-Learning to develop a model representing the combination of operations o

and their relationships, we henceforth refer to this technique as *QENAS: Q-Learning for Efficient Neural Architecture Search.* Below, we describe the two approaches we considered.

4.1 HalfQENAS

We want to measure the quality of a search space with respect to not only the individual operations o, but also the relationship between each pair $(o_i, o_j) \in \mathcal{E}$. For this purpose, we use a two-dimensional action space $\mathcal{A}$. The resulting action space is of the order $\mathcal{E} \times \mathcal{E}$.

$$Q(a) := Q(\cdot, (o_i, o_j))$$

When constructing networks, the controller iteratively selected operation layers o and stochastically applied skip connections from all previous layers. Previously, the LSTM was used to produce a probability distribution π from which to sample these new layers o. In this method, we opt to directly sample using Q-table rows. Note that architectures A are reduced to operation sets with undirected relationships between each pair (o_i, o_j). However, the previously mentioned LSTM is still retained to sample skip connections. We henceforth refer to this as HalfQENAS and provide the pseudocode in Algorithm 2. In sampling one operation $\hat{o}$ for A, we provide HalfQENAS-Sampling with the current Q-table, a partial architecture A'. The cumulative vector C sums the values v, which are the (possibly already soft-maxed) Q-values for each operation. C is further soft-maxed to produce the probability distribution π from which to sample a new operation. And finally, the LSTM makes skip connections between all previous layers $o \in A'$ to $\hat{o}$.

Algorithm 2 Sampling o from the Q-Table in Half QENAS

```
procedure HalfQENAS-Sampling(Q, A', sandwich)
    D ← dim(Q)
    C ← 0 ∈ ℝ^D
    for (o, s) ∈ A' do
        v ← Q[o]
        if sandwich then
            v ← softmax(v)
        C ← C + v
    π ← softmax(C)
    return π
```

The Q-table stores the most *extreme* (either minimum or maximum) performance per operation pair (o_i, o_j). Precisely, $Q((o_i, o_j))$ is equal to the most extreme reward R for any architecture A that includes both o_i and o_j. For example, in a search space $\mathcal{E}_x$ of 3 operations, consider the Q-table below. $Q_{1,2} = 0.63$ represents the best reward $R^*_{1,2}$ achieved by any network A that includes both o_1 and o_2. Q is therefore necessarily symmetric.

$$Q((o_i, o_j)) := \max_{\{A | o_i, o_j \in A\}} R \qquad Q_{\mathcal{E}_x} = \begin{bmatrix} 0.87 & 0.63 & 0.77 \\ 0.63 & 0.84 & 0.84 \\ 0.77 & 0.84 & 0.88 \end{bmatrix} \quad \text{(example)}$$

As with ENAS, our training step entails sampling architectures A and training shared CNN parameters ω. Similarly, REINFORCE is used to train the controller, but instead of

training the LSTM, we now update the Q-table. This is done by using an element-wise or vectorized Bellman-style update function f, which updates Q-values to any more extreme values. Both minimum and maximum are options for f to bootstrap Q. See Algorithm 3.

Algorithm 3 Updating the Q-Table in Half QENAS

procedure HalfQENAS-Update(Q, A, $R \in \mathbb{R}$, $f \in \{\min, \max\}$)
  for $o_i \in \text{set}(A)$ **do**
    for $o_j \in \text{set}(A)$ **do**
      $Q((o_i, o_j)) \leftarrow f(Q((o_i, o_j)), R)$
  return Q

4.2 Full Q-Learning

However, neural network architectures are best described as a directed computation graph. And previously, the Q-table is half-represented (and in-fact symmetric) and the LSTM is half-trained (refer to our codebase). Close inspection of HalfQENAS reveals a decorrelated LSTM whose learned decision-making about skip connections become less meaningful. Furthermore, it's inefficient to have to train both a Q-table and a half-baked LSTM.

To remedy this dilemma, we remove the LSTM entirely, and employ an augmented Q-table as a Q-Learning-based controller. We can now incorporate directed 0-1 relationships between operations o_i, o_j. We let the tuple represent (o_i, o_j, c) the fact that operations $o_i, o_j \in A$ coexist and that o_i has a feed-forward connection to o_j if and only if c is true. The $Q(a) := Q(\cdot, (o_i, o_j, c))$ table (now a tensor, not matrix) can be rewritten with an extra inner-most binary dimension to reflect this. This action space is in the order of $\mathcal{E} \times \mathcal{E} \times 2$.

Algorithm 4 Sampling o from the Q-Table in Full QENAS

procedure FullQENAS-Sampling(Q, A', sandwich)
  $D \leftarrow \dim(Q)$
  $C \leftarrow \mathbf{0} \in \mathbb{R}^D$
  for $(o, s) \in A'$ **do**
    $\mathbf{v} \leftarrow Q[o]$
    if sandwich **then**
      $v \leftarrow \text{softmax}(\mathbf{v})$
    $C \leftarrow C + \mathbf{v}$
  $\pi \leftarrow \text{softmax}(C)$
  $\hat{o} \leftarrow o \sim \pi$
  $\hat{s} \leftarrow []$
  for $(o, s) \in A'$ **do**
    $\mathbf{v} \leftarrow Q[o][\hat{o}]$
    $\hat{c} \leftarrow c \sim \pi_c(\mathbf{v})$
    append($\hat{s}, \hat{c}$)
  append($A', (\hat{o}, \hat{s})$)
  return A'

Accordingly, with the LSTM removed, the Q-table sampling process now includes both the operations selected and feed-forward connections made. The procedure for sampling operations hasn't changed, but instead of returning the probability distribution π, we directly sample $\hat{o} \sim \pi$, and then use it to further sample feed-forward relationships from previous operations (o, s). Logits for these relationships are captured in $\mathbf{v}$, and used in $\hat{c} \sim \pi_c(\mathbf{v})$ to sample whether or not to make a skip connection, where $\hat{c} \in \{0, 1\}$. And finally, because this FullQENAS-Sampling procedure (described in Algorithm 4) covers the entire sampling process, the updated partial architecture A' is returned.

Algorithm 5 Updating the Q-Table in Half QENAS

procedure FullQENAS-Update(Q, A, $R \in \mathbb{R}$, $f \in \{\min, \max\}$)
  for $(o, s) \in \text{set}(A)$ **do**
    if o isn't the first operation **then**
      $Q((o_{\text{prev}}, o, 1)) \leftarrow f(Q((o_{\text{prev}}, o, 1)), R)$ # OPTIONAL
    for $h, s_h \in \text{enumerate}(s)$ **do**
      $Q((A[h].o, o, s_h)) \leftarrow f(Q((A[h].o, o, s_h)), R)$
  return Q

Similarly, the process for learning the Q-table will also consider directed edges to represent relationships. For every operation layer o, we first update $Q((o_{\text{prev}}, o, c = 1))$ to account for the direct connection from the previous layer o_{prev} (if it exists); this is optional. We then enumerate through all the previous relationships $s_h \in s$ to update values in the Q-table corresponding to the 0-1 feed-forward relationship. See Algorithm 5.

5. Experiments

We had initially attempted to defend against one-shot poisoning (**?**). Unfortunately, because of the effectiveness of such poisoning attacks, the shared CNN parameters ω were spoiled beyond repair by exploding gradients that produce `NaN`'s. One technique we had attempted to employ was a mechanism to reset ω every time these attacks produced randomly guessing networks beyond repair, from **?**.

Our experiments were therefore set-up to defend against conventional search space poisoning (SSP) (Wu et al., 2021). As before, the task is CIFAR-10 classification. We had chosen our previous experiments with the dropout operation, as dropout was the single most poisonous but blatant operation we could inject into a search space $\mathcal{E}$. The basic experiments carried out by Wu et al. (2021) involved adding $|\mathcal{E}|$ bad operations as $\mathcal{P}$ into $\mathcal{E}$ such that $|\hat{\mathcal{E}}| = 2|\mathcal{E}|$. Note here that $\hat{\mathcal{E}}$ is a multiset, which allows for duplicated elements.

$$\mathcal{E} := \left\{ \begin{array}{cc} \text{3x3 Convs}, & \text{3x3 Separable Conv}, \\ \text{5x5 Conv}, & \text{5x5 Separable Conv}, \\ \text{Max Pool}, & \text{Avg Pool} \end{array} \right\}$$

$$\hat{\mathcal{E}_0} := \mathcal{E}_0 \cup 6\{\text{Dropout}(p = 1.0)\}$$

However, we hypothesized that these experiments would require an unreasonable runtime to produce meaningful results. Instead, we opt to add just a single dropout operation that brings the search space $\hat{\mathcal{E}_1}$ to size $|\hat{\mathcal{E}_1}| = |\mathcal{E}| + 1$.

$$\hat{\mathcal{E}_1} := \mathcal{E}_0 \cup \{\text{Dropout}(p = 1.0)\} = \left\{ \begin{array}{cc} \text{3x3 Convs}, & \text{3x3 Separable Conv}, \\ \text{5x5 Conv}, & \text{5x5 Separable Conv}, \\ \text{Max Pool}, & \text{Avg Pool}, \\ \text{Dropout}(p = 1.0) & \end{array} \right\}$$

5.1 Ablations

As we had multiple options when building this Q-learning system, we conducted the following experiments all on $\hat{\mathcal{E}_1}$ for 300 epochs on NVIDIA Quadro RTX 5000/8000 GPUs. Each experiment's controller was trained with the REINFORCE algorithm.

1. Baseline LSTM-based controller.
2. HalfQENAS controller with extrema $f := \max$; another with $f := \min$.
3. FullQENAS controller with extrema $f := \max$; another with $f := \min$.

FullQENAS will likely outperform HalfQENAS in both accuracy and efficiency due to the latter's use of a decorrelated LSTM for skip connection sampling. However, we were interested to see how Q-Learning in general would compare to the baseline LSTM.

6. Results

Right off the bat, it's clear that in terms of classification accuracy, the LSTM-based controller from the ENAS baseline far outperforms our Q-Learning methods. In fact, it achieves nearly the same performance with $\hat{\mathcal{E}_1}$ as it does $\mathcal{E}_0$ (Wu et al., 2021). This is somewhat expected because we were under the impression that our three-dimensional Q-table would retain more information from the relationships between operations in each architecture A.

Experiment	Op Sampling	Skip Sampling	f	Val Error	Test Error
Baseline ENAS	LSTM	LSTM	N/A	21.75%	24.87%
HalfQENAS max	Q-table	LSTM	max	54.27%	54.06%
HalfQENAS min	Q-table	LSTM	min	52.87%	56.97%
FullQENAS max	Q-table	Q-table	max	**39.26**%	**42.92**%
FullQENAS min	Q-table	Q-table	min	46.29%	51.47%

Table 1: Child network classification errors on CIFAR-10 (lower is better).

Between our Q-learning methods however, it appears that we were right that aiding Q-Learning with the LSTM for connection sampling in HalfQENAS resulted in lower performance than FullQENAS due to the decorrelated LSTM. Another observation of note was that using extreme function $f := \max$ tends to be more effective than $f := \min$. We suspect this is because it doesn't mean as much how bad the worst performance of an operation o or pair (o_i, o_j) is because there will always stochastically bad outliers. While the same is technically true for the better outliers, they are a better estimation of the true performance over time with lower variance near the upper performance bound.

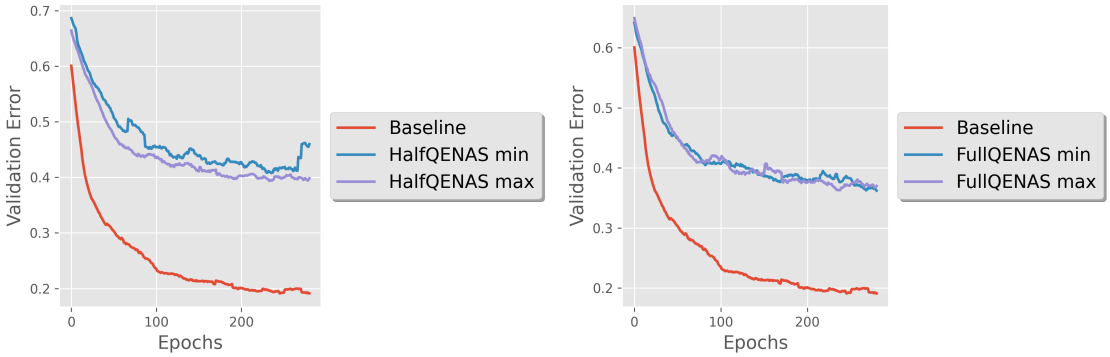

Figure 1: Comparing HalfQENAS and FullQENAS to the ENAS baseline.

6.1 Computational Cost

While the overall outcome disappointingly favours the LSTM-based controller from the ENAS baseline, Q-Learning did show promise in computational efficiency. All of our ablations performed faster than the baseline, slightly in controller training, and significantly in shared CNN parameters ω training. In both steps, it appears that the use of the LSTM directly slows down the sampling and update processes. The minor differences between the extrema in each variation were likely just due to randomness.

Experiment	Op Sampling	Skip Sampling	CNN Training Speed	Controller Training Speed
Baseline ENAS	LSTM	LSTM	4.78 ± 1.13 it/s	1.57 ± 0.41 it/s
HalfQENAS max	Q-table	LSTM	6.43 ± 0.79 it/s	1.97 ± 0.14 it/s
HalfQENAS min	Q-table	LSTM	6.23 ± 0.85 it/s	1.92 ± 0.11 it/s
FullQENAS max	Q-table	Q-table	10.40 ± 1.03 it/s	2.46 ± 0.08 it/s
FullQENAS min	Q-table	Q-table	9.72 ± 1.42 it/s	2.49 ± 0.17 it/s

Table 2: Training step speeds as iterations per second (higher is better).

7. Conclusion

Unfortunately, we were not able to outperform the ENAS baseline with the HalfQENAS or FullQENAS variations; perhaps simple Q-Learning is just not powerful enough. However, we still believe that RL-based controllers can still bring good performance to the table, especially in the context of one-shot poisoning (**?**) and more complex tasks. At the very least, we have shown that Q-Learning-based methods are much faster than the LSTM architecture when training both the shared parameters and controller.

7.1 Source Code

- FullQENAS: `https://github.com/rusbridger/ENAS-Experiments/tree/qenas`
- HalfQENAS: Commit `6fe528f892b3f4a0f98bd4cef136677bbd945b4f`
- $\hat{\mathcal{E}}_1$: `github.com/rusbridger/enas_poisoning/blob/master/P3_001_100.py`

References

Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389*, 2012.

Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. Adversarial attacks and defences: A survey. *arXiv preprint arXiv:1810.00069*, 2018.

Liang-Chieh Chen, Maxwell D Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jonathon Shlens. Searching for efficient multi-scale architectures for dense image prediction. *arXiv preprint arXiv:1809.04184*, 2018.

Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. Neural architecture search: A survey. *J. Mach. Learn. Res.*, 20(55):1–21, 2019.

Ignazio Gallo, Gabriele Magistrali, Nicola Landro, and Riccardo La Grassa. Improving the efficient neural architecture search via rewarding modifications. In *2020 35th International Conference on Image and Vision Computing New Zealand (IVCNZ)*, pages 1–6, 2020. doi: 10.1109/IVCNZ51579.2020.9290732.

Minghao Guo, Yuzhe Yang, Rui Xu, Ziwei Liu, and Dahua Lin. When nas meets robustness: In search of robust architectures against adversarial attacks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 631–640, 2020.

Attila Nagy and Ábel Boros. Improving the sample-efficiency of neural architecture search with reinforcement learning, 2021.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning*, pages 4095–4104. PMLR, 2018.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

Nayan Saxena, Robert Wu, and Rohan Jain. Towards one shot search space poisoning in neural architecture search (student abstract). In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 13043–13044, 2022.

Mihaela van der Schaar. Automl and interpretability: Powering the machine learning revolution in healthcare. In *Proceedings of the 2020 ACM-IMS on Foundations of Data Science Conference*, pages 1–1, 2020.

Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. A survey on neural architecture search. *arXiv preprint arXiv:1905.01392*, 2019.

Robert Wu, Nayan Saxena, and Rohan Jain. Poisoning the search space in neural architecture search. In *Workshop on Adversarial Machine Learning, 38th International Conference on Machine Learning*. ICML, 2021.

Huang Xiao, Battista Biggio, Gavin Brown, Giorgio Fumera, Claudia Eckert, and Fabio Roli. Is feature selection secure against training data poisoning? In *International Conference on Machine Learning*, pages 1689–1698. PMLR, 2015.

Chaofei Yang, Qing Wu, Hai Li, and Yiran Chen. Generative poisoning attack method against neural networks. *arXiv preprint arXiv:1703.01340*, 2017.

Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. *arXiv preprint arXiv:1902.08142*, 2019.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.