Neural Ensemble Search for Uncertainty Estimation and Dataset Shift

Sheheryar Zaidi[1*] Arber Zela[2*] Thomas Elsken[2]
Chris Holmes[1] Frank Hutter[2,3] Yee Whye Teh[1]

[1]University of Oxford [2]University of Freiburg [3]Bosch Center for Artificial Intelligence

February 2021

Quantifying Uncertainty in Deep Learning

Why is uncertainty important?

- **Predictive uncertainty** can be for instance the output label together with the confidence of that prediction in classification

Quantifying Uncertainty in Deep Learning

Why is uncertainty important?

- **Predictive uncertainty** can be for instance the output label together with the confidence of that prediction in classification
- Good uncertainty estimates quantify how much we **can trust our model's predictions**

Quantifying Uncertainty in Deep Learning

Why is uncertainty important?

- **Predictive uncertainty** can be for instance the output label together with the confidence of that prediction in classification
- Good uncertainty estimates quantify how much we **can trust our model's predictions**

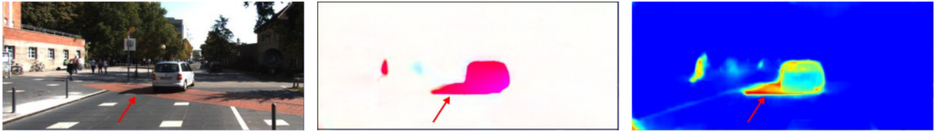


Figure: (from [Ilg et al. 2018]) Optical flow and its uncertainty estimation.

Quantifying Uncertainty in Deep Learning

Why is uncertainty important?

- **Predictive uncertainty** can be for instance the output label together with the confidence of that prediction in classification
- Good uncertainty estimates quantify how much we **can trust our model's predictions**
- Some applications where uncertainty quantification is important are:
  - Cost-sensitive decision making (healthcare e.g. medical imaging; self-driving cars; robotics)
  - Dealing with distribution shift (Feature skew between train and test sets; test inputs do not belong to any of the training classes)
  - Safe exploration in RL, etc.

Calibration and Robustness to dataset shift

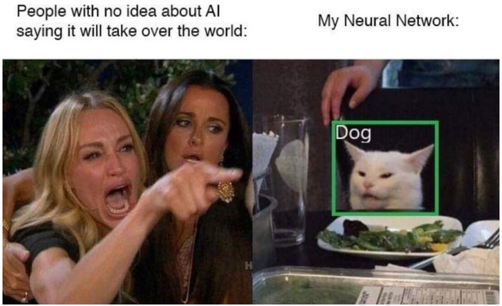
Can we trust our model predictions?

- Ideally, a reliable model should yield maximum uncertainty when faced with an out-of-distribution (OOD) example, i.e. it knows what it doesn't know.

Calibration and Robustness to dataset shift

Can we trust our model predictions?

- Ideally, a reliable model should yield maximum uncertainty when faced with an out-of-distribution (OOD) example, i.e. it knows what it doesn't know.
  - Usually neural networks are **not well-calibrated** and **overconfident** when they should not be.



Calibration and Robustness to dataset shift

Can we trust our model predictions?

- Ideally, a reliable model should yield maximum uncertainty when faced with an out-of-distribution (OOD) example, i.e. it knows what it doesn't know.
  - Usually neural networks are **not well-calibrated** and **overconfident** when they should not be.

- Calibration tells us how well the predicted confidence (*probability of correctness*) of the model aligns with the observed accuracy (*frequency of correctness*).
  - E.g. in classification: if the correct predicted class was always with 80% probability, then a perfectly calibrated system would imply that on 80% of the examples it predicted the true class.

Ensembles of neural networks

Starting point

- *Ensembles* of networks are commonly used to boost performance.

Ensembles of neural networks

Starting point

- *Ensembles* of networks are commonly used to boost performance.
- Recent interest in ensembles has been due to their strong *predictive uncertainty estimation* and *robustness to distributional shift*.

Ensembles of neural networks

Starting point

- *Ensembles* of networks are commonly used to boost performance.
- Recent interest in ensembles has been due to their strong *predictive uncertainty estimation* and *robustness to distributional shift*.
- Diversity among the *base learners*' predictions is believed to be key for strong ensembles.

- Notation: f_θ is a network with weights θ, and $\ell(f_\theta(\boldsymbol{x}), y)$ is the loss for $(\boldsymbol{x}, y)$. Define the ensemble of M networks $f_{\theta_1}, \ldots, f_{\theta_M}$ by $F(\boldsymbol{x}) = \frac{1}{M}\sum_{i=1}^{M} f_{\theta_i}(\boldsymbol{x})$.

Ensembles of neural networks

On diversity in ensembles

- Notation: f_θ is a network with weights θ, and $\ell(f_\theta(\boldsymbol{x}), y)$ is the loss for $(\boldsymbol{x}, y)$. Define the ensemble of M networks $f_{\theta_1}, \dots, f_{\theta_M}$ by $F(\boldsymbol{x}) = \frac{1}{M}\sum_{i=1}^{M} f_{\theta_i}(\boldsymbol{x})$.
- **Average base learner loss:** $\frac{1}{M}\sum_{i=1}^{M} \ell(f_{\theta_i}(\boldsymbol{x}), y)$.

- Notation: f_θ is a network with weights θ, and $\ell(f_\theta(\boldsymbol{x}), y)$ is the loss for $(\boldsymbol{x}, y)$. Define the ensemble of M networks $f_{\theta_1}, \dots, f_{\theta_M}$ by $F(\boldsymbol{x}) = \frac{1}{M}\sum_{i=1}^{M} f_{\theta_i}(\boldsymbol{x})$.
- **Average base learner loss:** $\frac{1}{M}\sum_{i=1}^{M} \ell(f_{\theta_i}(\boldsymbol{x}), y)$.
- **Oracle ensemble:** given $f_{\theta_1}, \dots, f_{\theta_M}$, the oracle ensemble F_{OE} is defined as

$$F_{\mathsf{OE}}(\boldsymbol{x}) = f_{\theta_k}(\boldsymbol{x}), \quad \text{where} \quad k \in \operatorname*{argmin}_i \ell(f_{\theta_i}(\boldsymbol{x}), y).$$

- Notation: f_θ is a network with weights θ, and $\ell(f_\theta(\boldsymbol{x}), y)$ is the loss for $(\boldsymbol{x}, y)$. Define the ensemble of M networks $f_{\theta_1}, \ldots, f_{\theta_M}$ by $F(\boldsymbol{x}) = \frac{1}{M}\sum_{i=1}^{M} f_{\theta_i}(\boldsymbol{x})$.
- **Average base learner loss:** $\frac{1}{M}\sum_{i=1}^{M} \ell(f_{\theta_i}(\boldsymbol{x}), y)$.
- **Oracle ensemble:** given $f_{\theta_1}, \ldots, f_{\theta_M}$, the oracle ensemble F_{OE} is defined as

$$F_{\mathsf{OE}}(\boldsymbol{x}) = f_{\theta_k}(\boldsymbol{x}), \quad \text{where} \quad k \in \operatorname*{argmin}_i \ell(f_{\theta_i}(\boldsymbol{x}), y).$$

- As a rule of thumb, *small oracle ensemble loss indicates more diverse base learner predictions*.

Proposition

Suppose ℓ is negative log-likelihood (NLL). Then, the oracle ensemble loss, ensemble loss, and average base learner loss satisfy the following inequality:

$$\ell(F_{OE}(\boldsymbol{x}), y) \leq \ell(F(\boldsymbol{x}), y) \leq \frac{1}{M} \sum_{i=1}^{M} \ell(f_{\theta_i}(\boldsymbol{x}), y).$$

Proof.

Direct application of Jensen's inequality for the right inequality and definition of oracle ensemble for the left one. □

Ensembles of neural networks

Deep Ensembles

- Typical approaches, such as deep ensembles [Lakshminarayanan et al. 2017], only ensembles predictions coming from the same *fixed* architecture as follows:
  1. Independently train multiple copies of a fixed architecture with random initializations.
  2. Create an ensemble by averaging outputs, i.e. predicted distribution over the classes (in the case of classification).

Ensembles of neural networks

Deep Ensembles

- Typical approaches, such as deep ensembles [Lakshminarayanan et al. 2017], only ensembles predictions coming from the same *fixed* architecture

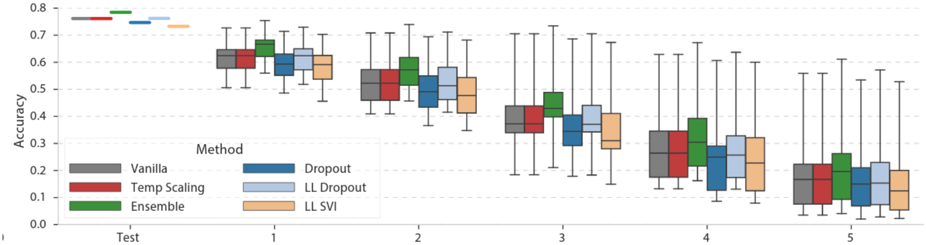


Figure: Test accuracy on ImageNet (from [Ovadia et al. 2019])

Ensembles of neural networks
Deep Ensembles

- Typical approaches, such as deep ensembles [Lakshminarayanan et al. 2017], only ensembles predictions coming from the same *fixed* architecture
- **Why only use a fixed architecture?** Would ensembling different architectures result in higher diversity among the ensemble predictions?

Ensembles of neural networks

Deep Ensembles

- Typical approaches, such as deep ensembles [Lakshminarayanan et al. 2017], only ensembles predictions coming from the same *fixed* architecture
- **Why only use a fixed architecture?** Would ensembling different architectures result in higher diversity among the ensemble predictions?
  - → We propose a procedure to automatically construct ensembles of varying architectures over *complex*, state-of-the-art architectural search spaces.

Ensembles of neural networks

Deep Ensembles

- Typical approaches, such as deep ensembles [Lakshminarayanan et al. 2017], only ensembles predictions coming from the same *fixed* architecture
- **Why only use a fixed architecture?** Would ensembling different architectures result in higher diversity among the ensemble predictions?
  - → We propose a procedure to automatically construct ensembles of varying architectures over *complex*, state-of-the-art architectural search spaces.
  - → Varying the base learner architectures increases diversity → ensembles have better predictive performance and uncertainty, in-distribution and during shift.

Varying vs. fixed base learner architectures

Visualizing base learner predictions using t-SNE on CIFAR-10

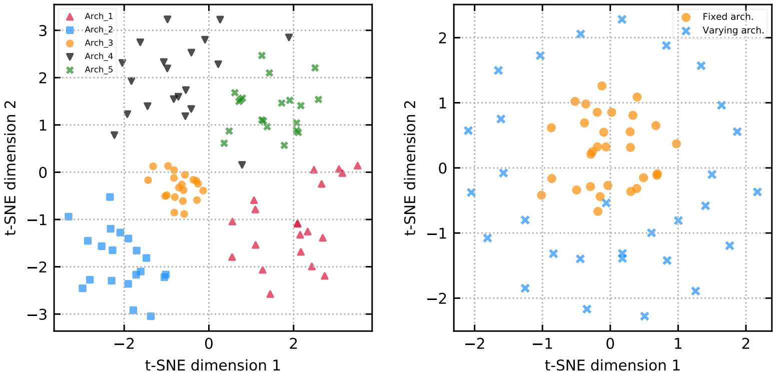


Figure: **Left:** Predictions of 5 different archs, each trained with 20 different inits. **Right:** Predictions of base learners in an ensemble with varying archs (found using NES) vs. fixed arch (deep ensemble of optimized arch).

Neural Ensemble Search

Problem formulation

- Let $\mathcal{L}(f, \mathcal{D}) = \sum_{(\boldsymbol{x},y)\in\mathcal{D}} \ell(f(\boldsymbol{x}), y)$ be the loss of f over dataset $\mathcal{D}$ and let `Ensemble` be the function which maps a set of base learners $\{f_1, \ldots, f_M\}$ to the ensemble $F = \frac{1}{M}\sum_{i=1}^{M} f_i$. A NES algorithm aims to solve the following optimization problem:

$$\min_{\alpha_1,\ldots,\alpha_M\in\mathcal{A}} \mathcal{L}\left(\texttt{Ensemble}(f_{\theta_1,\alpha_1}, \ldots, f_{\theta_M,\alpha_M}), \mathcal{D}_{\text{val}}\right)$$

$$\text{s.t.} \quad \theta_i \in \operatorname*{argmin}_{\theta} \mathcal{L}(f_{\theta,\alpha_i}, \mathcal{D}_{\text{train}}) \qquad \text{for } i = 1, \ldots, M$$

where $\mathcal{A}$ is the architecture search space.

Neural Ensemble Search

Problem formulation

- Let $\mathcal{L}(f, \mathcal{D}) = \sum_{(\boldsymbol{x},y)\in\mathcal{D}} \ell(f(\boldsymbol{x}), y)$ be the loss of f over dataset $\mathcal{D}$ and let `Ensemble` be the function which maps a set of base learners $\{f_1, \ldots, f_M\}$ to the ensemble $F = \frac{1}{M}\sum_{i=1}^{M} f_i$. A NES algorithm aims to solve the following optimization problem:

$$\min_{\alpha_1,\ldots,\alpha_M\in\mathcal{A}} \mathcal{L}\left(\texttt{Ensemble}(f_{\theta_1,\alpha_1}, \ldots, f_{\theta_M,\alpha_M}), \mathcal{D}_{\text{val}}\right)$$

$$\text{s.t.} \quad \theta_i \in \operatorname*{argmin}_{\theta} \mathcal{L}(f_{\theta,\alpha_i}, \mathcal{D}_{\text{train}}) \qquad \text{for } i = 1, \ldots, M$$

  where $\mathcal{A}$ is the architecture search space.
- The search space size is effectively $\mathcal{A}^M$, compared to it being $\mathcal{A}$ in typical NAS.

Neural Ensemble Search

General approach

- Our approach for finding base learner architectures that optimize ensemble performance consists of two steps. Let $f_{\theta,\alpha}$ denote a network with arch α and weights θ. Computational budget denoted by K and ensemble size by M.

Neural Ensemble Search

General approach

- Our approach for finding base learner architectures that optimize ensemble performance consists of two steps. Let $f_{\theta,\alpha}$ denote a network with arch α and weights θ. Computational budget denoted by K and ensemble size by M.
   1. **Pool building**: build a *pool* $\mathcal{P} = \{f_{\theta_1,\alpha_1}, \ldots, f_{\theta_K,\alpha_K}\}$ of size K consisting of potential base learners, where each f_{θ_i,α_i} is a network trained independently on $\mathcal{D}_{\text{train}}$.

Neural Ensemble Search

General approach

- Our approach for finding base learner architectures that optimize ensemble performance consists of two steps. Let $f_{\theta,\alpha}$ denote a network with arch α and weights θ. Computational budget denoted by K and ensemble size by M.
  1. **Pool building**: build a *pool* $\mathcal{P} = \{f_{\theta_1,\alpha_1}, \dots, f_{\theta_K,\alpha_K}\}$ of size K consisting of potential base learners, where each f_{θ_i,α_i} is a network trained independently on $\mathcal{D}_{\text{train}}$.
  2. **Ensemble selection**: select M base learners $f_{\theta_1^*,\alpha_1^*}, \dots, f_{\theta_M^*,\alpha_M^*}$ from $\mathcal{P}$ to form an ensemble which minimizes loss on $\mathcal{D}_{\text{val}}$. (We assume $K \geq M$.)

Neural Ensemble Search

General approach

- Our approach for finding base learner architectures that optimize ensemble performance consists of two steps. Let $f_{\theta,\alpha}$ denote a network with arch α and weights θ. Computational budget denoted by K and ensemble size by M.
  1. **Pool building**: build a *pool* $\mathcal{P} = \{f_{\theta_1,\alpha_1}, \ldots, f_{\theta_K,\alpha_K}\}$ of size K consisting of potential base learners, where each f_{θ_i,α_i} is a network trained independently on $\mathcal{D}_{\text{train}}$.
  2. **Ensemble selection**: select M base learners $f_{\theta_1^*,\alpha_1^*}, \ldots, f_{\theta_M^*,\alpha_M^*}$ from $\mathcal{P}$ to form an ensemble which minimizes loss on $\mathcal{D}_{\text{val}}$. (We assume $K \geq M$.)
- For step 2, we use forward step-wise selection without replacement: given pool $\mathcal{P}$, start with an empty ensemble and add to it the network from $\mathcal{P}$ which minimizes ensemble loss on $\mathcal{D}_{\text{val}}$. We repeat this without replacement until the ensemble is of size M [Caruana et al., 2004].

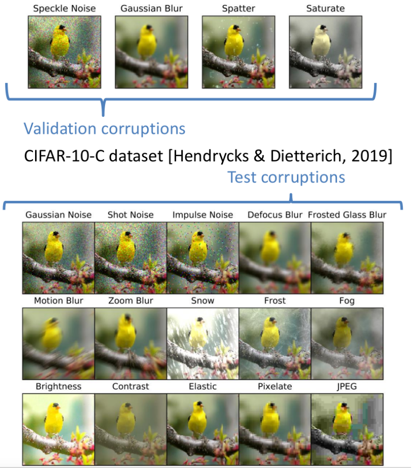
Neural Ensemble Search

General approach

- Our approach for finding base learner architectures that optimize ensemble performance consists of two steps. Let $f_{\theta,\alpha}$ denote a network with arch α and weights θ. Computational budget denoted by K and ensemble size by M.
  1. **Pool building**: build a *pool* $\mathcal{P} = \{f_{\theta_1,\alpha_1}, \ldots, f_{\theta_K,\alpha_K}\}$ of size K consisting of potential base learners, where each f_{θ_i,α_i} is a network trained independently on $\mathcal{D}_{\text{train}}$.
  2. **Ensemble selection**: select M base learners $f_{\theta_1^*,\alpha_1^*}, \ldots, f_{\theta_M^*,\alpha_M^*}$ from $\mathcal{P}$ to form an ensemble which minimizes loss on $\mathcal{D}_{\text{val}}$. (We assume $K \geq M$.)
- For step 2, we use forward step-wise selection without replacement: given pool $\mathcal{P}$, start with an empty ensemble and add to it the network from $\mathcal{P}$ which minimizes ensemble loss on $\mathcal{D}_{\text{val}}$. We repeat this without replacement until the ensemble is of size M [Caruana et al., 2004].
- Later we discuss two options for pool building in step 1.

- We assume that at test time, the data will contain a distributional shift(s) wrt training data. The shift(s) is assumed to be unknown at training time.

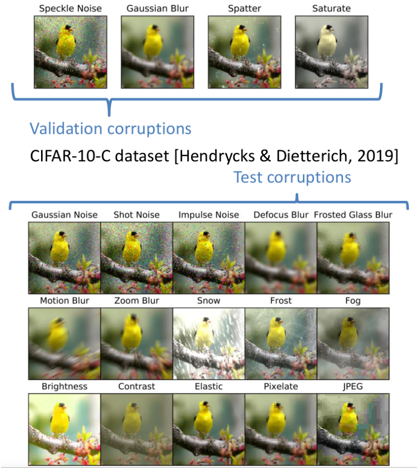
- We assume that at test time, the data will contain a distributional shift(s) wrt training data. The shift(s) is assumed to be unknown at training time.
- We consider the case where a validation dataset with *validation* shift(s) is available.



Neural Ensemble Search

Ensemble Adaptation to Dataset Shift

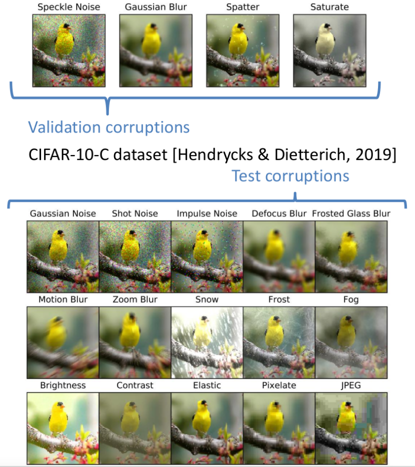
- We assume that at test time, the data will contain a distributional shift(s) wrt training data. The shift(s) is assumed to be unknown at training time.
- We consider the case where a validation dataset with *validation* shift(s) is available.
- To adapt the ensembles to shift, simply replace $\mathcal{D}_{\text{val}}$ with the shifted validation dataset $\mathcal{D}_{\text{val}}^{\text{shift}}$.



Neural Ensemble Search

Ensemble Adaptation to Dataset Shift

- We assume that at test time, the data will contain a distributional shift(s) wrt training data. The shift(s) is assumed to be unknown at training time.
- We consider the case where a validation dataset with *validation* shift(s) is available.
- To adapt the ensembles to shift, simply replace $\mathcal{D}_{\text{val}}$ with the shifted validation dataset $\mathcal{D}_{\text{val}}^{\text{shift}}$.
- Roughly (and heuristically), diversity in ensembles is particularly useful during shift. Using a shifted validation set allows NES algorithms to "consider" what happens to baselearners when they're used during shift (and are likely to fail).



- **NES-RS** is a simple random search (RS) based approach: we build the pool by sampling K architectures uniformly at random.

Neural Ensemble Search

NES-RS: with random search

- **NES-RS** is a simple random search (RS) based approach: we build the pool by sampling K architectures uniformly at random.
- Motivation: in NAS, RS is a competitive baseline on well-designed architecture search spaces [Li & Talwalkar 2019]. Applying ensemble selection to the pool of randomly sampled archs is then a simple way to exploit diversity among varying archs.

Neural Ensemble Search

NES-RS: with random search

Algorithm 1: NES with Random Search

Data: Search space $\mathcal{A}$; ensemble size M; comp. budget K; $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}$.

1 Sample K architectures $\alpha_1, \ldots, \alpha_K$ independently and uniformly from $\mathcal{A}$.

2 Train each architecture α_i using $\mathcal{D}_{\text{train}}$, yielding a pool of networks $\mathcal{P} = \{f_{\theta_1,\alpha_1}, \ldots, f_{\theta_K,\alpha_K}\}$.

3 Select base learners $\{f_{\theta_1^*,\alpha_1^*}, \ldots, f_{\theta_M^*,\alpha_M^*}\} = \texttt{ForwardSelect}(\mathcal{P}, \mathcal{D}_{\text{val}}, M)$ by forward step-wise selection without replacement.

4 **return** ensemble $\texttt{Ensemble}(f_{\theta_1^*,\alpha_1^*}, \ldots, f_{\theta_M^*,\alpha_M^*})$

Figure: NES-RS. $f_{\theta,\alpha}$ is a network with *weights* θ and *architecture* α.

- NES-RE uses another approach for pool building inspired by regularized evolution [Real et al., 2018]. The arch search space is explored by *evolving a population of architectures* till a budget K is reached.

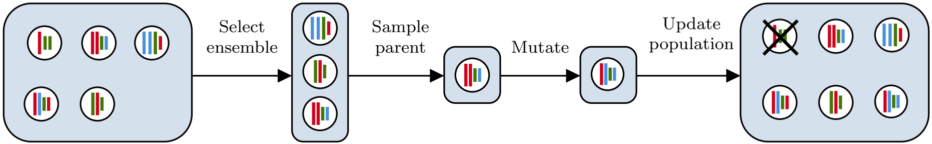


Figure: One iteration of NES-RE. Network architectures are represented as colored bars of different lengths illustrating different layers and widths. The pool returned is the set of *all architectures evaluated.*

Neural Ensemble Search

NES-RE: with Regularized Evolution

Algorithm 2: NES with Regularized Evolution

Data: Search space $\mathcal{A}$; ensemble size M; comp. budget K; $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}$; population size P;

Neural Ensemble Search

NES-RE: with Regularized Evolution

Algorithm 2: NES with Regularized Evolution

Data: Search space $\mathcal{A}$; ensemble size M; comp. budget K; $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}$; population size P; number of parent candidates m.

Sample P architectures $\alpha_1, \ldots, \alpha_P$ independently and uniformly from $\mathcal{A}$.

Train each architecture α_i using $\mathcal{D}_{\text{train}}$, and initialize $\mathfrak{p} = \mathcal{P} = \{f_{\theta_1,\alpha_1}, \ldots, f_{\theta_P,\alpha_P}\}$.

Neural Ensemble Search

NES-RE: with Regularized Evolution

Algorithm 2: NES with Regularized Evolution

Data: Search space $\mathcal{A}$; ensemble size M; comp. budget K; $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}$; population size P; number of parent candidates m.

Sample P architectures $\alpha_1, \ldots, \alpha_P$ independently and uniformly from $\mathcal{A}$.
Train each architecture α_i using $\mathcal{D}_{\text{train}}$, and initialize $\mathfrak{p} = \mathcal{P} = \{f_{\theta_1,\alpha_1}, \ldots, f_{\theta_P,\alpha_P}\}$.
while $|\mathcal{P}| < K$ **do**
    if $\mathcal{D}_{\text{val}}^{\text{shift}}$ *is available* **then**
        $\mathcal{D}_{\text{val}} \leftarrow \mathcal{D} \sim \{\mathcal{D}_{\text{val}}, \mathcal{D}_{\text{val}}^{\text{shift}}\}$ `// randomly pick between clean & shifted`
    Select m parent candidates $\{f_{\widetilde{\theta}_1,\widetilde{\alpha}_1}, \ldots, f_{\widetilde{\theta}_m,\widetilde{\alpha}_m}\} =$ `ForwardSelect`$(\mathfrak{p}, \mathcal{D}_{\text{val}}, m)$.
    Sample uniformly a parent architecture α from $\{\widetilde{\alpha}_1, \ldots, \widetilde{\alpha}_m\}$. `//` α `stays in` $\mathfrak{p}$`.`
    Apply mutation to α, yielding child architecture β.
    Train β using $\mathcal{D}_{\text{train}}$ and add the trained network $f_{\theta,\beta}$ to $\mathfrak{p}$ and $\mathcal{P}$.

Algorithm 2: NES with Regularized Evolution

Data: Search space $\mathcal{A}$; ensemble size M; comp. budget K; $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}$; population size P; number of parent candidates m.

Sample P architectures $\alpha_1, \dots, \alpha_P$ independently and uniformly from $\mathcal{A}$.
Train each architecture α_i using $\mathcal{D}_{\text{train}}$, and initialize $\mathfrak{p} = \mathcal{P} = \{f_{\theta_1,\alpha_1}, \dots, f_{\theta_P,\alpha_P}\}$.
while $|\mathcal{P}| < K$ **do**
    if $\mathcal{D}_{\text{val}}^{\text{shift}}$ *is available* **then**
        $\mathcal{D}_{\text{val}} \leftarrow \mathcal{D} \sim \{\mathcal{D}_{\text{val}}, \mathcal{D}_{\text{val}}^{\text{shift}}\}$ `// randomly pick between clean & shifted`
    Select m parent candidates $\{f_{\widetilde{\theta}_1,\widetilde{\alpha}_1}, \dots, f_{\widetilde{\theta}_m,\widetilde{\alpha}_m}\} =$ `ForwardSelect`$(\mathfrak{p}, \mathcal{D}_{\text{val}}, m)$.
    Sample uniformly a parent architecture α from $\{\widetilde{\alpha}_1, \dots, \widetilde{\alpha}_m\}$. `//` α `stays in` $\mathfrak{p}$.
    Apply mutation to α, yielding child architecture β.
    Train β using $\mathcal{D}_{\text{train}}$ and add the trained network $f_{\theta,\beta}$ to $\mathfrak{p}$ and $\mathcal{P}$.
    Remove the oldest member in $\mathfrak{p}$. `// as done in RE (Real et al., 2019).`

Neural Ensemble Search

NES-RE: with Regularized Evolution

Algorithm 2: NES with Regularized Evolution

Data: Search space $\mathcal{A}$; ensemble size M; comp. budget K; $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}$; population size P; number of parent candidates m.

Sample P architectures $\alpha_1, \dots, \alpha_P$ independently and uniformly from $\mathcal{A}$.
Train each architecture α_i using $\mathcal{D}_{\text{train}}$, and initialize $\mathfrak{p} = \mathcal{P} = \{f_{\theta_1,\alpha_1}, \dots, f_{\theta_P,\alpha_P}\}$.
while $|\mathcal{P}| < K$ **do**
    if $\mathcal{D}_{\text{val}}^{\text{shift}}$ *is available* **then**
        $\mathcal{D}_{\text{val}} \leftarrow \mathcal{D} \sim \{\mathcal{D}_{\text{val}}, \mathcal{D}_{\text{val}}^{\text{shift}}\}$ `// randomly pick between clean & shifted`
    Select m parent candidates $\{f_{\widetilde{\theta}_1,\widetilde{\alpha}_1}, \dots, f_{\widetilde{\theta}_m,\widetilde{\alpha}_m}\}$ = `ForwardSelect`$(\mathfrak{p}, \mathcal{D}_{\text{val}}, m)$.
    Sample uniformly a parent architecture α from $\{\widetilde{\alpha}_1, \dots, \widetilde{\alpha}_m\}$.   `//` α `stays in` $\mathfrak{p}$.
    Apply mutation to α, yielding child architecture β.
    Train β using $\mathcal{D}_{\text{train}}$ and add the trained network $f_{\theta,\beta}$ to $\mathfrak{p}$ and $\mathcal{P}$.
    Remove the oldest member in $\mathfrak{p}$.   `// as done in RE (Real et al., 2019).`
Select base learners $\{f_{\theta_1^*,\alpha_1^*}, \dots, f_{\theta_M^*,\alpha_M^*}\}$ = `ForwardSelect`$(\mathcal{P}, \mathcal{D}_{\text{val}}, M)$ by forward step-wise selection without replacement.
return ensemble `Ensemble`$(f_{\theta_1^*,\alpha_1^*}, \dots, f_{\theta_M^*,\alpha_M^*})$

Experimental results

On the DARTS [Liu et al. 2019] search space; Fashion-MNIST

- We compare ensembles found by NES with the baseline of deep ensembles composed of a fixed, optimized architecture; the optimized arch is either DARTS, AmoebaNet or optimized by RS.

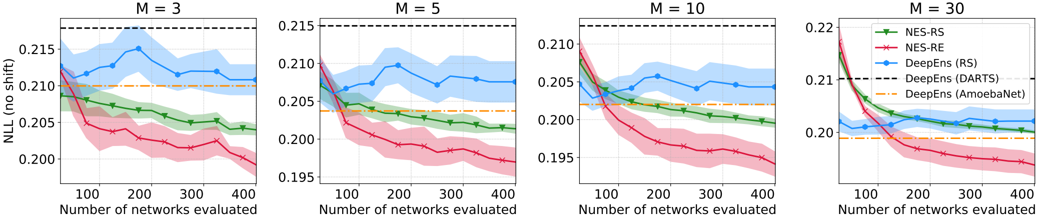
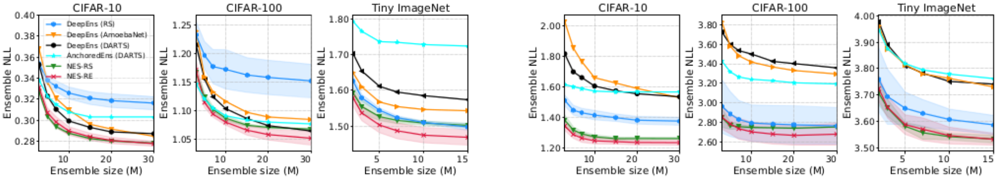


Figure: Negative log-likelihood achieved by ensembles on test data. Note that AmoebaNet arch is deeper than all other methods shown. M is ensemble size.

Experimental results

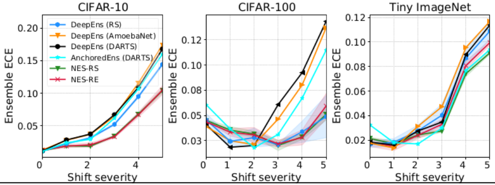
On the DARTS [Liu et al. 2019] search space: CIFAR-10/100, Tiny ImageNet

- NLL vs. ensemble size after 400 iterations of NES:



(a) No data shift (b) Dataset shift: severity 5 (out of 5)

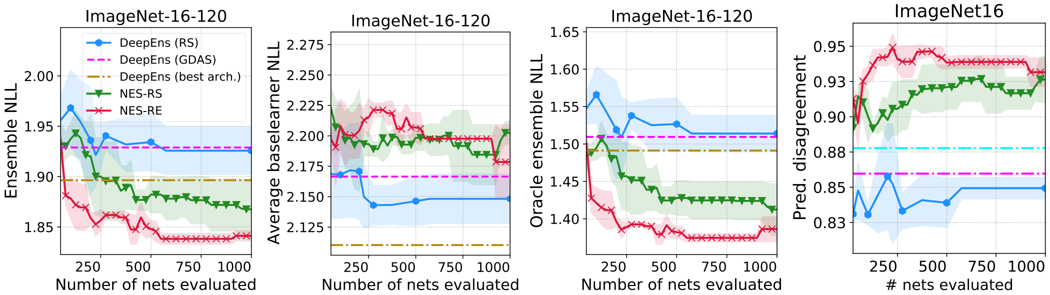
- Expected Calibration Error (ECE) vs. shift severity level after 400 iterations of NES ($M = 10$).



Experimental results

Results on NAS-Bench-201 [Dong & Yang 2020]: CIFAR-10/100 and ImageNet-16-120

By yielding more diverse base learners (lower *oracle NLL and higher predictive disagreement*), NES outperforms deep ensembles of a fixed architecture, even though the latter contains better individual base learners (lower *average base learner NLL*).



Experimental results

Results on NAS-Bench-201 [Dong & Yang 2020]: CIFAR-10/100 and ImageNet-16-120

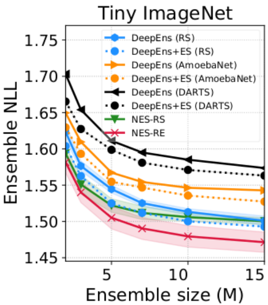
NES outperforms DeepEns (best arch.) with up to 14 classification error (mean $\pm 95\%$ confidence interval of 3 runs) percentage points.

Dataset	Shift Severity	Classif. error (%), $\mathcal{A}$ = NAS-Bench-201 search space				
		DeepEns (GDAS)	DeepEns (best arch.)	DeepEns (RS)	NES-RS	NES-RE
CIFAR-10	0	8.4	**7.2**	7.8$_{\pm 0.2}$	7.7$_{\pm 0.1}$	7.6$_{\pm 0.1}$
	3	28.7	27.1	28.3$_{\pm 0.3}$	**22.0**$_{\pm 0.2}$	22.5$_{\pm 0.1}$
	5	47.8	46.3	37.1$_{\pm 0.0}$	**32.5**$_{\pm 0.2}$	33.0$_{\pm 0.5}$
CIFAR-100	0	29.9	26.4	26.3$_{\pm 0.4}$	**23.3**$_{\pm 0.3}$	**23.5**$_{\pm 0.2}$
	3	60.3	54.5	57.0$_{\pm 0.9}$	**46.6**$_{\pm 0.3}$	**46.7**$_{\pm 0.5}$
	5	75.3	69.9	64.5$_{\pm 0.0}$	**59.7**$_{\pm 0.2}$	60.0$_{\pm 0.6}$
ImageNet-16-120	0	49.9	49.9	50.5$_{\pm 0.6}$	**48.1**$_{\pm 1.0}$	**47.9**$_{\pm 0.4}$

Experimental results

Computational cost and further results

- NES is typically better then deep ensembles with the seeds selected from a pool via `ForwardSelect`.
- The primary computational cost in NES is training K nets to form the pool.
- NES merges the 2-step procedure of finding a good architecture and then creating deep ensembles.



Method	Cost (# nets trained)	
	Arch.	Ensemble
DeepEns (DARTS)	32	10
DeepEns + ES (DARTS)	32	200
DeepEns (AmoebaNet)	25200	10
DeepEns + ES (AmoebaNet)	25200	200
DeepEns (RS)	200	10
DeepEns + ES (RS)	200	200
NES-RS	200	
NES-RE	200	

Summary and future directions

- Varying architectures of base learners improves diversity.

Summary and future directions

- Varying architectures of base learners improves diversity.
- We propose two algorithms to effectively search for these architectures.

Summary and future directions

- Varying architectures of base learners improves diversity.
- We propose two algorithms to effectively search for these architectures.
- NES searches for more diverse ensembles without ever explicitly defining diversity.

Summary and future directions

- Varying architectures of base learners improves diversity.
- We propose two algorithms to effectively search for these architectures.
- NES searches for more diverse ensembles without ever explicitly defining diversity.
- We show improved performance and better calibration for in-distribution and shifted data.

Summary and future directions

- Varying architectures of base learners improves diversity.
- We propose two algorithms to effectively search for these architectures.
- NES searches for more diverse ensembles without ever explicitly defining diversity.
- We show improved performance and better calibration for in-distribution and shifted data.

In the future:

- NES-BO: NES via Bayesian Optimization.

Summary and future directions

- Varying architectures of base learners improves diversity.
- We propose two algorithms to effectively search for these architectures.
- NES searches for more diverse ensembles without ever explicitly defining diversity.
- We show improved performance and better calibration for in-distribution and shifted data.

In the future:

- NES-BO: NES via Bayesian Optimization.
- Differentiable NES.

Summary and future directions

- Varying architectures of base learners improves diversity.
- We propose two algorithms to effectively search for these architectures.
- NES searches for more diverse ensembles without ever explicitly defining diversity.
- We show improved performance and better calibration for in-distribution and shifted data.

In the future:

- NES-BO: NES via Bayesian Optimization.
- Differentiable NES.
- NES in joint NAS and HPO spaces.