# Accelerating CNN Hyperparameter Tuning with TPE and Statistical Early Stopping

Alison Wong
University of Washington
Seattle, WA
asnwong@uw.edu

## Abstract

*Hyperparameter optimization (HPO) plays a crucial role in training high-performing deep learning models, but existing methods such as Random Search, Hyperband, and Tree-structured Parzen Estimators (TPE) often rely on predefined iteration budgets, leading to inefficient use of computational resources if suboptimal configurations are continuously evaluated after an optimal hyperparameter configuration is evident. This project presents Auto-TPE, an automatic termination method for training convolutional neural networks (CNNs) using TPE. Auto-TPE dynamically terminates the HPO process once the potential for a better hyperparameter configuration is statistically unlikely. Auto-TPE's performance will be compared to other benchmarked methods: Random Search, Hyperband, and TPE.*

## 1. Introduction

Hyperparameter optimization (HPO) is an important step in training convolution neural networks (CNNs). By optimizing the hyperparameters correctly, we observed an improvement in CNN classification accuracy of up to 6% [9]. However, this comes with its own difficulty as CNNs have many hyperparameters such as number of filters in each convolution layer, kernal size, pool size, dropout rate, learning rate, batch size, and many more. CNN training is computationally expensive, and each HPO trial involves significant cost in both time and resources. As such, efficient methods for selecting optimal configurations are essential.

Traditional HPO approaches such as grid search and random search operate under fixed budgets and treat each configuration independently, not taking prior experiments into account [4]. Additionally, evaluating five hyperparameters with just ten values each would result in $10^5 = 10 \times 10^4$ combinations. Evaluating each configuration quickly becomes computationally infeasible, making brute-force approaches impractical, especially for larger models.

Bayesian optimization (BO) has become a popular framework for improving HPO efficiency. BO techniques model the objective function using a surrogate model, such as a Gaussian Process or TPE, and choose new hyperparameter configurations to evaluate by maximizing an acquisition function, such as Expected Improvement (EI) or Upper Confidence Bound (UCB) [4].

Despite these advances, a major limitation of most HPO strategies remains: they depend on predefined iteration budgets or threshold-based early stopping, which may continue evaluating suboptimal configurations long after the optimal one is evident. Methods such as Hyperband and Successive Halving mitigate this by allocating more resources to promising configurations, but they still require setting global resource limits and do not guarantee termination when a clear winner emerges [4].

This project proposes Auto-TPE: a TPE framework with an automatic early stopping criterion to terminate the HPO loop once additional evaluations are unlikely to yield meaningful improvement 3.1. This method expands on Makarova et al. (2022) [7] stopping rule for BO with Gaussian Processes (GP) when the upper bound on simple regret is less than the standard deviation of the cross-validation estimate. It stops when the potential improvement is smaller than the statistical noise.

Formally, the search terminates when:

$$\min_{\gamma \in \mathcal{G}_t} \mathrm{UCB}_t(\gamma) \;-\; \min_{\gamma \in \Gamma} \mathrm{LCB}_t(\gamma) \;<\; \sqrt{\mathrm{Var}[\hat{f}(\gamma_t^*)]},$$

where:

$$\mathrm{UCB}_t(\gamma) = \mu_t(\gamma) + \sqrt{\beta_t} \cdot \sigma_t(\gamma),$$
$$\mathrm{LCB}_t(\gamma) = \mu_t(\gamma) - \sqrt{\beta_t} \cdot \sigma_t(\gamma).$$

Here, $\mu_t(\gamma)$ and $\sigma_t(\gamma)$ denote the mean and standard deviation predicted by the surrogate model at configuration $\gamma$, and $\beta_t$ is a confidence parameter that controls exploration.

The right-hand side approximates the variance of the $k$-fold cross-validation estimator at the current best configuration $\gamma_t^*$:

$$\begin{aligned}\mathrm{Var}[\hat{f}(\gamma_t^*)] \;\approx&\\ &\left(\frac{1}{k} + \frac{n_{val}}{n_{train}}\right) s_{\mathrm{cv}}^2(\gamma_t^*)\\ &\approx\; 0.21 \cdot s_{\mathrm{cv}}^2(\gamma_t^*) \quad \text{(for 10-fold CV)}\end{aligned}$$

where $s_{\mathrm{cv}}^2(\gamma_t^*)$ is the sample variance of the cross-validation losses, $n_{val}$ is the size of the validation fold, and $n_{train}$ the size of the training remainder.

### 1.1. Bayesian Optimization (BO)

BO treats hyperparameter tuning as a sequential decision problem by leveraging previous evaluations to find good hyperparameter configurations in fewer iterations than RS or HB [4].

**Surrogate model.** At step $t$, $\hat{f}_t(x)$ probabilistically approximates the expensive objective $f(x)$ (the CNN's validation loss).

**Acquisition function.** This computationally cheap function guides the search (explore or exploit) and decides which hyperparameter configuration to test next.

$$x_{t+1} \;=\; \arg\max_{x \in \mathcal{X}} a_t(x\,;\, \hat{f}_t).$$

The true loss $y_{t+1} = f(x_{t+1})$ is evaluated and used to optimize the surrogate model until the training budget ends.

Common choices are a Gaussian process surrogate with Expected Improvement (EI) as the acquisition function,

$$a_{\mathrm{EI}}(x) = \mathbb{E}\big[\max(0,\, y^\star - \hat{f}(x))\big],$$

where $y^\star$ is the incumbent best.

A key limitation of BO is that the acquisition function sets the search space early, so the model can miss an important feature and get stuck at a local optima [8].

## 2. Related Work

HPO is an active field of research within the deep learning community. As models become larger and more complicated, we need better HPO methods to reduce training time and computation resources.

Many recent BO variants focus on improving scalability and robustness while reducing resource usage. Multi-fidelity BO [4] extends classical BO by evaluating configurations on cheaper, lower-fidelity approximations (e.g. fewer epochs, lower resolution) before committing full resources. BOHB (2018) [3] integrates Bayesian search with Hyperband's resource allocation strategy, achieving strong performance under tight budgets.

Parallel BO addresses BO's traditional sequential bottleneck by enabling asynchronous or batch evaluations. Kandasamy et al. (2018) [5] use Thompson Sampling to generate parallel queries with minimal coordination overhead, making BO viable in large distributed systems.

Beyond model-based approaches, Zoph and Le (2016) [11] demonstrated that neural networks themselves can be used to learn optimization strategies, forming the basis for neural architecture search (NAS) and controller-based HPO.

However, Xu et al. (2025) [10] recently challenged the necessity of increasingly complex BO variants. They argued that a standard Gaussian Process with carefully tuned length scales is sufficient for high-dimensional BO, questioning the practical gains of newer models.

Despite methodological progress, comparing HPO strategies remains difficult. As Feurer and Hutter [4] noted, existing studies vary widely in reported metrics (validation / test accuracy, runtime, number of iterations) and often lack reproducible baselines. Open-source frameworks such as Optuna, HyperOpt, and SMAC make state-of-the-art methods accessible, yet no single library unifies all basic building blocks, impeding direct method-to-method comparisons.

## 3. Methods

### 3.1. HPO Methods

Auto-TPE's performance will be compared to three other benchmarked HPO algorithms: Random Search, Hyperband, and TPE. They all use different optimization methods: brute-force, bandit-based algorithm, blackbox BO methods.

**Random Search (RS)** draws each configuration independently from a user-defined range in the search space [2]. RS aims to improve the curse of dimensionality that grid search faces, where a full factorial design is performed to find the best hyperparameter configuration.

Bergstra & Bengio [2] show that sampling $n$ points at random covers at least one configuration within the top $10\%$ of the search space with probability

$$\Pr\big(\text{success}\big) = 1 - (1 - 0.10)^n.$$

Setting $\Pr \geq\, 0.90$ yields $n \geq 22$. This means that RS with at least 22 trials will find the closest-to-optimal region of hyperparameter configurations if it occupies at least $10\%$ of the hyperparameter space grid. Also, unlike black-box optimization methods, it will not get stuck in local optima.

There are no assumptions in RS and it prioritizes exploration. RS scales trivially across workers and forms a neutral baseline against which more sophisticated methods can be judged.

**Hyperband (HB)** is a bandit-based algorithm selection method that accelerates RS by adaptively distributing computational resources. [6] Firstly, a predefined range is set. It starts with a small initial budget and tests all hyperparameters for that budget. HB then repeatedly applies *Successive Halving* - discarding the worst performers and multiplying the training budget of the survivors until a few configurations are trained at full fidelity.

HB makes minimal assumptions, allows for exploration while also ensuring exploitation by keeping the best configuration until the end.

**Tree-structured Parzen Estimator (TPE).** The TPE is a BO model-based sequential optimizer [1].

Given a set of observed hyperparameter configurations $\{(x_i, y_i)\}_{i=1}^n$, where $x_i$ is a hyperparameter configuration and $y_i$ is the corresponding validation loss:

TPE splits the observations into:

- $L = \{x_i \mid y_i < y^*\}$ (good configurations)
- $G = \{x_i \mid y_i \geq y^*\}$ (bad configurations)

where $y^*$ is a predefined threshold (e.g., the $\gamma$-quantile of observed losses).

Then, kernel density estimators (KDEs) are fitted for each hyperparameter dimension separately:

$$l(x) = p(x \mid y < y^*), \quad g(x) = p(x \mid y \geq y^*)$$

We sample candidates $x$ from $l(x)$ (good density) and then score each candidate using the acquisition function:

$$\text{Score}(x) = \frac{l(x)}{g(x)} = \prod_{j=1}^{d} \frac{l_j(x_j)}{g_j(x_j)}$$

assuming independent KDEs per hyperparameter dimension $j$.

The next configuration is selected by:

$$x^* = \arg\max_x \frac{l(x)}{g(x)}$$

and $x^*$ is evaluated on the objective function and update the history. This repeats until a stopping criterion is met.

**Auto-TPE (Proposed Method).** We propose an automatic early stopping criterion for TPE that stops the search when the potential for further improvement is smaller than the inherent statistical uncertainty in validation performance. Specifically, the optimization terminates when the difference between the mean cross validation (CV) losses of the best and second-best configurations is smaller than the standard deviation of the CV loss of the best configuration. This approach avoids over-searching once progress becomes statistically identical, suggesting that if the gap is small enough, the configuration is close to optimal.

Since TPE does not provide explicit uncertainty estimates, our method adapts Makarova's rule [7] to operate on point estimates from cross-validation losses.

Adapted Stopping Criterion (TPE):

$$\hat{f}(\gamma_{(2)}) - \hat{f}(\gamma_{(1)}) \; < \; \sqrt{0.21 \cdot s^2_{\text{cv}}(\gamma_{(1)})},$$

where:

$$\hat{f}(\gamma_{(1)}) = \min_{\gamma \in \mathcal{H}_t} \hat{f}(\gamma),$$
$$\gamma_{(2)} = \arg \min_{\gamma \in \mathcal{H}_t \setminus \{\gamma_{(1)}\}} \hat{f}(\gamma),$$

$\gamma$ is the hyperparameter configuration, and $\mathcal{H}_t$ is the set of all configurations evaluated up to iteration $t$.

We approximate the variance of the CV mean with $s^2_{\text{cv}}(\gamma_{(1)})$ - the sample variance of the $k$-fold CV losses for the best configuration. The constant 0.21 reflects the variance factor under a 10-fold CV:

$$\text{Var}[\hat{f}(\gamma_{(1)})] \approx \left(\frac{1}{k} + \frac{n_{val}}{n_{train}}\right) s^2_{\text{cv}}(\gamma_{(1)}) \approx 0.21 \cdot s^2_{\text{cv}}(\gamma_{(1)}).$$

**Algorithm 1:** Auto-TPE

```
Input: search space $\mathcal{S}$, max trials $T_{\max}$, $k$-fold CV,
       min trials $T_{\min}$
Result: Best configuration $\gamma^\star$
$R \leftarrow \emptyset$        // history of ($\gamma$, $\hat{f}$, Var($\gamma$))
for $t = 1, \ldots, T_{\max}$ do
    1.;
    $\gamma_t \leftarrow$ TPE($R, \mathcal{S}$)
    2.;
    Evaluate $\gamma_t$ with $k$-fold CV
        Compute mean loss $\hat{f}_t$ and variance $s_t^2$;
        $_t \leftarrow 0.21\, s_t^2$
    3.;
    Append $(\gamma_t, \hat{f}_t,_t)$ to $R$
    4.;
    if $|R| \geq T_{\min}$ then
        Sort $R$ by $\hat{f}$; let $L^\star$ and $L^{(2)}$ be the best and
          second–best means; let $^\star$ be the variance of
          the best.
        if $L^{(2)} - L^\star < \sqrt{^\star}$ then
            break                    // early stop
return configuration in $R$ with smallest $\hat{f}$
```

## 4. Experiments

### 4.1. CNN architecture

| Stage | Layer sequence |
|---|---|
| Input | Image $\rightarrow$ Rescaling ($\times 1/255$) |
| 1 | $2\times2$ Conv (90 filters) $\rightarrow$ ReLU $\rightarrow$ BatchNorm |
| 2 | $5\times5$ Conv (120 filters) $\rightarrow$ ReLU $\rightarrow$ BatchNorm $\rightarrow$ $3\times3$ Max-Pool |
| 3 | $2\times2$ Conv (125 filters) $\rightarrow$ ReLU $\rightarrow$ BatchNorm $\rightarrow$ $3\times3$ Max-Pool $\rightarrow$ Dropout ($p$=0.1) |
| 4 | Flatten $\rightarrow$ Affine$^*$ $\rightarrow$ LeakyReLU$^*$ $\rightarrow$ BatchNorm $\rightarrow$ Dropout$^*$ |
| 5 | Affine (softmax output) |

Table 1. CNN architecture used across all HPO methods. (*: apart of the hyperparameter search space).

### 4.2. Experiment Setup

- **Search space.** Each method has the same HPO search space. All discrete sets to keep comparison simple. Focus on HPO for the last FC layer in the CNN network.

| Hyperparameter | Symbol | Candidate values |
|---|---|---|
| Hidden units | $h$ | 20, 40, 60 |
| Bias constant | $b_0$ | 0, 0.20, 0.40, 0.60 |
| Leaky-ReLU slope | $\lambda$ | 0, 0.15, 0.30, 0.60 |
| Dropout rate | $d$ | 0, 0.10, 0.20, 0.80 |

Table 2. Search space for the FC layer.

- **Architecture & training loop.** The CNN is trained with $E$=5 epochs and batch size 6 with AdamW (learning rate 0.0001, weight-decay 0.01). The maximum number of iterations / trials is set to 22.

- **Dataset size.** Shapes (squares/circles/crosses): image size: 32×32×1, training set: 4800 images, testing set: 1200 images.

- **Data split.** Deterministic data split and global seed for fairness and reproducibility.

- **Hardware.** Single NVIDIA T4 GPU.

## 5. Results

### 5.1. Comparison with Benchmarked Methods

| Method | Affine Layer | Dropout | ReLU Leak | Bias Init |
|---|---|---|---|---|
| Random Search | 60 | 0.20 | 0.60 | 0.00 |
| Hyperband | 20 | 0.20 | 0.60 | 0.20 |
| TPE | 20 | 0.10 | 0.15 | 0.40 |
| Auto-TPE | 40 | 0.20 | 0.30 | 0.60 |

Table 3. Best hyperparameter configuration for each HPO method.

| Method | Validation Accuracy | Time (seconds) |
|---|---|---|
| Random Search | **0.9983** | 2933.99 |
| Hyperband | 0.9967 | 3816.87 |
| TPE | 0.9967 | 4703.47 |
| Auto-TPE | 0.9908 | **2532.84** |

Table 4. Validation accuracy for best configuration and total runtime for each HPO method.

### 5.2. Auto-TPE (Proposed Method)

**After Trial 3 (t = 3)**

$$\begin{aligned}
\text{Best mean\_loss} &= 0.037385 \quad (\text{t = 2; var} = 0.0002122) \\
\text{Second-best mean\_loss} &= 0.113709 \quad (\text{t = 3}) \\
\text{Regret} &= 0.113709 - 0.037385 = 0.076324 \\
\text{Noise} &= \sqrt{0.21 \times 0.0002122} \approx 0.00668 \\
\Rightarrow \quad & 0.07632 > 0.00668 \quad \Rightarrow \text{Continue}
\end{aligned}$$

**After Trial 4 (t = 4)**

$$\begin{aligned}
\text{Best mean\_loss} &= 0.037385 \quad (\text{t = 2; var} = 0.0002122) \\
\text{Second-best mean\_loss} &= 0.042632 \quad (\text{t = 4}) \\
\text{Regret} &= 0.042632 - 0.037385 = 0.005247 \\
\text{Noise} &= \sqrt{0.21 \times 0.0002122} \approx 0.00668 \\
\Rightarrow \quad & 0.00525 < 0.00668 \quad \Rightarrow \text{Stop}
\end{aligned}$$

When we applied Auto-TPE 3.1 to the CNN classification task for shapes. In total, four complete trials were needed before the stopping criterion was satisfied. Trial 1's mean CV loss was relatively large (0.11520), the method waited until a genuinely good loss (0.0374) was found, and only then compared the top two.

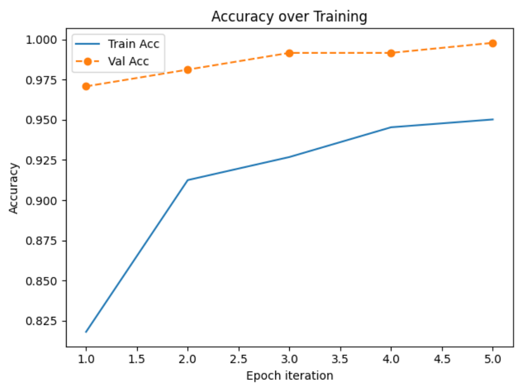

Figure 1. Training and Validation Accuracy by Epochs of Auto-TPE's best hyperparameter configuration.

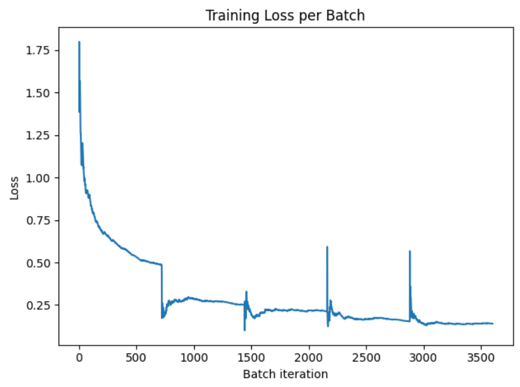

Figure 2. Training Loss per Batch of Auto-TPE's best hyperparameter configuration.

The training accuracy is lower than the validation accuracy, which is likely due to the dropout layers during CNN training. Although the training accuracy is still rising, the training loss reached a plateau. This means that epochs = 5 is a good value as additional epochs could lead to overfitting.

## 6. Discussion

Auto-TPE allowed for early stopping when TPE already found a strong CNN configuration, which was 46% faster than vanilla TPE.

Auto-TPE did not outperform the other methods in accuracy, however, it was the fastest out of all methods. This presents a trade-off between time and accuracy. Random Search achieved the best validation accuracy (0.9983), but required the largest budget.

If the goal is obtaining the highest accuracy and the computational resources are not limited, Random Search or vanilla TPE is still preferable. However, if run-time or GPU budget is limited, Auto-TPE delivers a near-optimal model in significantly less time and resources.

### 6.1. Limitations and Future Work

It could be possible that the best configuration has a large loss and the difference between the best and second-best mean CV loss is small. Thus, the Auto-TPE will stop even if we have not actually found a good configuration. To avoid this from happening, we can set a minimum threshold for the mean CV loss or set a minimum number of iterations before checking the stopping rule. This brings in another limitation, as choosing a threshold for mean CV loss requires domain knowledge and is another hyperparameter to worry about.

The current implementation of the Auto-TPE algorithm loads the entire training set into NumPy arrays to allow for K-fold splits. For larger datasets, this will take up a lot of memory. Thus, further optimizations can be implemented on the algorithm.

Using cross-validation comes with its limitations. If the validation data distribution is not representative of the test distribution. For example, if test images have different lighting or angles, the validation and test data distribution could differ. Thus, stopping based on CV noise could be biased towards the validation metric.

Another limitation of Auto-TPE is that it takes on the disadvantages of the underlying TPE algorithm. TPE assumes that each hyperparameter dimension is independent and models them with one-dimensional KDEs [1]. When there is a strong correlation between hyperparameters, this will lead to suboptimal exploration. Additionally, TPE could be dataset-specific. Thus, if TPE itself is not a good fit for a problem, Auto-TPE may not have optimal performance.

In our experiment, the architecture of convolution layers was fixed, creating a slight bias towards black-box model-based methods like TPE, as the network is already near convergence and is less likely to get stuck in local optima.

In the future, to improve generality, we can repeat our experiments with different datasets and/or hyperparameters in other layers of a CNN (given that higher computational resources are available). Additionally, increasing the size of the search space also allows us to see how the method performs in larger configurations.

## References

[1] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger, editors, *NIPS*, pages 2546–2554, 2011. 3, 5

[2] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012. 2

[3] Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale, 2018. 2

[4] Matthias Feurer and Frank Hutter. Hyperparameter optimization. In Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors, *Automated Machine Learning*, The Springer Series on Challenges in Machine Learning, pages 3–33. Springer, Cham, 2019. 1, 2

[5] Kirthevasan Kandasamy, Akshay Krishnamurthy, Jeff Schneider, and Barnabas Poczos. Parallelised bayesian optimisation via thompson sampling. In Amos Storkey and Fernando Perez-Cruz, editors, *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*, pages 133–142. PMLR, 09–11 Apr 2018. 2

[6] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018. 3

[7] Anastasia Makarova, Huibin Shen, Valerio Perrone, Aaron Klein, Jean Baptiste Faddoul, Andreas Krause, Matthias Seeger, and Cedric Archambeau. Automatic termination for hyperparameter optimization, 2022. 1, 3

[8] A. H. Victoria and G. Maragatham. Automatic tuning of hyperparameters using bayesian optimization. *Evolving Systems*, 12:217–223, Mar. 2021. Received 21 December 2019; Accepted 15 May 2020; Published 25 May 2020; Issue date March 2021. 2

[9] Mikolaj Wojciuka, Zaneta Swiderska-Chadaja, Krzysztof Siweka, and Arkadiusz Gertych. Improving classification accuracy of fine-tuned cnn models: Impact of hyperparameter optimization. *Heliyon*, 10(5):e26586, Mar. 2024. Received 3 December 2022; Revised 2 January 2024; Accepted 15 February 2024; Published online 22 February 2024; Issue date March 15, 2024; Open access under CC BY 4.0. 1

[10] Zhitong Xu, Haitao Wang, Jeff M Phillips, and Shandian Zhe. Standard gaussian process is all you need for high-dimensional bayesian optimization, 2025. 2

[11] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016. 2