International Baccalaureate Extended Essay

Computer Science

Comparing the Efficiency of Random Search and Tree-Structured Parzen Estimator Algorithms to Optimize Convolutional Neural Networks for Image Recognition

Session: May 2018

Word Count: 3994

Table of Contents

1. Research Introduction
2. Background
   - 2.1 Image Recognition
   - 2.2 Convolutional Neural Network (CNN)
   - 2.3 Hyperparameter Optimization
   - 2.4 Random Search Algorithm
   - 2.5 Tree-Structured Parzen Estimator Algorithm
3. Methodology to Compare Random Search and Tree-Structured Parzen Estimator
   - 3.1 Experiments Design
   - 3.2 Chosen Hyperparameters
   - 3.3 Libraries Descriptions
   - 3.4 Dataset Descriptions
4. Comparing Random Search and Tree-Structured Parzen Estimator for Simple CNN
   - 4.1 Architecture of Simple CNN
   - 4.2 Optimal Number of Convolutional Layers
   - 4.3 Optimal Batch Size
   - 4.4 Optimal Activation Functions
   - 4.5 Optimal Dropout Values
   - 4.6 Optimal Optimizer Method
   - 4.7 Optimal Values for all Chosen Hyperparameters
5. Comparing Random Search and Tree-Structured Parzen Estimator for Complex CNN
   - 5.1 Architecture of Complex CNN
   - 5.2 Optimal Number of Convolutional Layers
   - 5.3 Optimal Batch Size
   - 5.4 Optimal Activation Functions
   - 5.5 Optimal Dropout Values

5.6 Optimal Optimizer Method

5.7 Optimal Values for All Chosen Hyperparameters

6. Comparing the Efficiency of Random Search and Tree-Structured Parzen Estimator for all Chosen Hyperparameters in Simple and Complex CNNs

6.1 Accuracy

6.2 Loss

7. Conclusion

8. Limitations of this Paper

9. Further Investigation

10. Bibliography

11. Appendix

A. Source Code of Hyperparameter Optimization for Simple CNN

A.1 Optimal Number of Convolutional Layers

A.2 Optimal Batch Size

A.3 Optimal Activation Functions

A.4 Optimal Dropout Values

A.5 Optimal Optimizer Method

A.6 Optimal Values for All Chosen Hyperparameters

B. Source Code of Hyperparameter Optimization for Complex CNN

B.1 Optimal Number of Convolutional Layers

B.2 Optimal Batch Size

B.3 Optimal Activation Functions

B.4 Optimal Dropout Values

B.5 Optimal Optimizer Method

B.6 Optimal Values for All Chosen Hyperparameters

C. Accuracy and Loss of All Chosen Hyperparameters for Simple CNN

C.1 Random Search

C.2 Tree-Structured Parzen Estimator

D. Accuracy and Loss of All Chosen Hyperparameters for Complex CNN

D.1 Random Search

D.2 Tree-Structured Parzen Estimator

1. Research Introduction

In the increasingly sophisticated and technologically advanced world, image recognition has become a very popular technology that contributes to a wide variety of applications. One of the most effective approaches is convolutional neural network, which is a type of artificial neural network that uses biologically-inspired neurons and repeated convolutions to analyze visual imageries. For any dataset of images, convolutional neural networks with different hyperparameters have different performances. Two algorithms that optimize the hyperparameters are Random Search and Tree-Structured Parzen Estimator. This paper is an investigation on the effectiveness of Random Search and Tree-Structured Parzen Estimator to optimize convolutional neural networks for image recognition.

2. Background

2.1 Image Recognition

Image recognition refers to the process of computers to determine and recognize the image data as belonging to a set of categories[1]. As one of the most popular and classical problem in computer vision and image processing, it embodies a variety of more specific recognition tasks, such as object identification, detection, and segmentation[1]. The approaches and algorithms produced for image recognition contribute to a wide variety of modern applications, including content-based image retrieval, pose estimation, optical character recognition, and facial recognition.

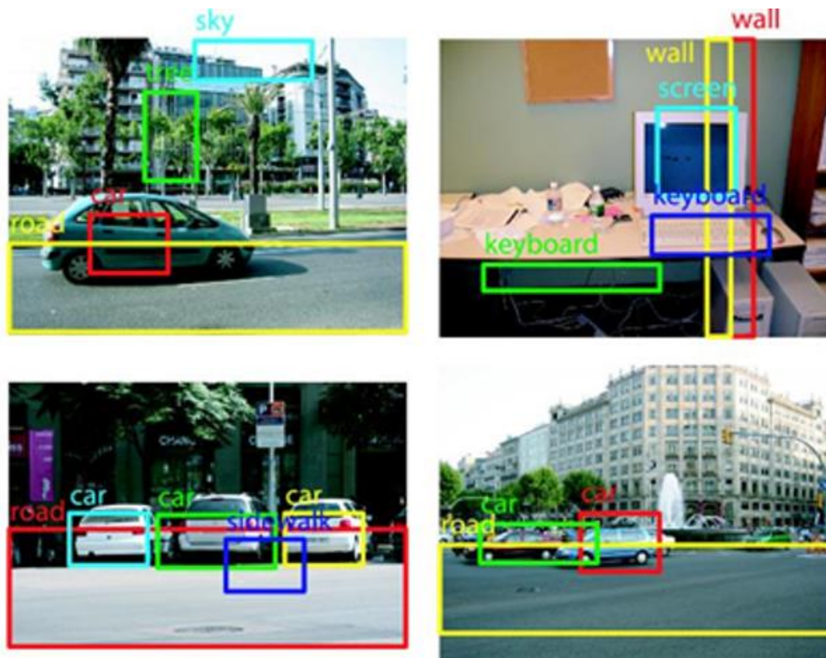

Figure 1 Image Recognition on Objects [2]

[1] Li, Fei-Fei. "Image Classification." *CS231n Convolutional Neural Networks for Visual Recognition*, Stanford University, cs231n.github.io/classification/.

[2] Russel, Bryan. "Object Recognition by Scene Alignment." *Object Recognition by Scene Alignment*, Bryan Russel, bryanrussell.org/projects/recognitionBySceneAlignment/index.html.

Currently, the algorithms that have the best performances for image recognition are based on convolutional neural networks, due to their highly effective handling of image data and efficient management of memory and storage space.

2.2 Convolutional Neural Network

The **convolutional neural network (CNN)** is a type of artificial neural network that has been successfully adapted to image analyzing and classifying tasks[3]. Its most prominent feature is convolution, which extracts features from a given image by stepping through the pixels of the input data[4]. The purpose of CNN in image classification is to categorize images into their specific classes by conducting a series of mathematical operations on the image data.

[3] Ujjwalkarn. "An Intuitive Explanation of Convolutional Neural Networks." *An Intuitive Explanation of Convolutional Neural Networks*, The Data Science Blog, 11 Aug. 2016, ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/.

[4] Ujjwalkarn. "An Intuitive Explanation of Convolutional Neural Networks." *An Intuitive Explanation of Convolutional Neural Networks*, The Data Science Blog, 11 Aug. 2016, ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/.

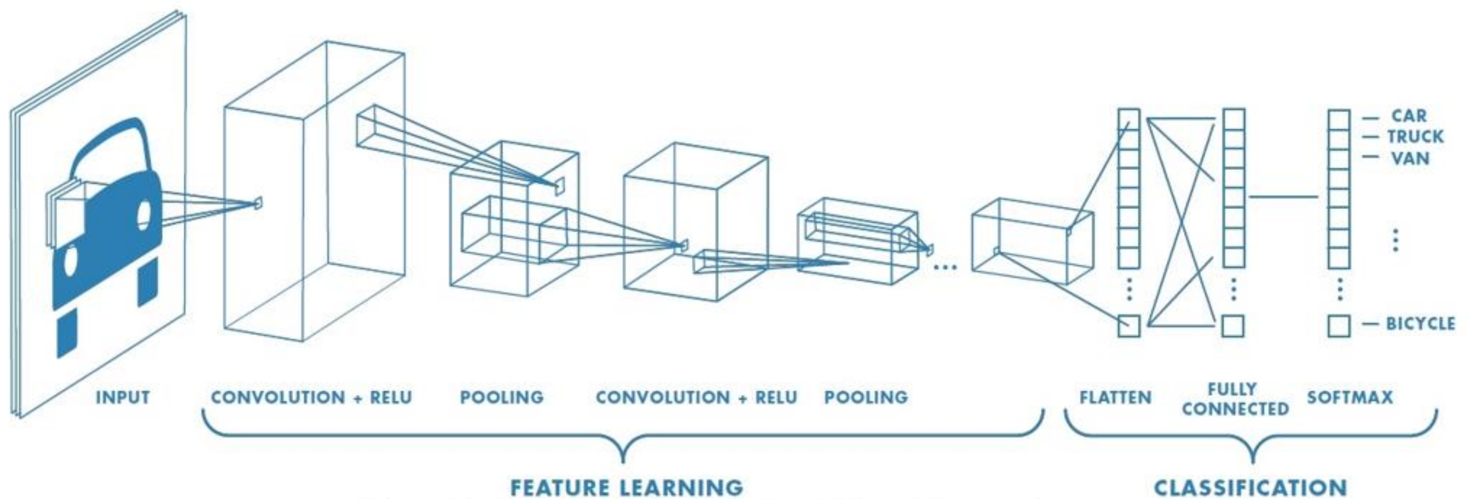

Figure 2 Layers of a Convolutional Neural Network

5

A CNN consists of a number of different layers, each specified for different tasks of image recognition. In the following descriptions, the image of a car is used as an example to explain the architecture of a CNN.

At the first layer, which is the **input** layer, the image of a car is fed to the CNN.

Within the **convolutional layers**, the pixel values of the car image are traversed by filters that search for different features of the car. The convolutions perform dot products between the values in the filters and the pixel values of the images to form new feature maps, which are inputted to the next layer[6]. Each feature map Y_k can be computed by

[5] "Convolutional Neural Network." *Convolutional Neural Network*, Mathlab, www.mathworks.com/discovery/convolutional-neural-network.html.

[6] Ujjwalkarn. "An Intuitive Explanation of Convolutional Neural Networks." *An Intuitive Explanation of Convolutional Neural Networks*, The Data Science Blog, 11 Aug. 2016, ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/.

$$Y_k = f(W_k \cdot x)$$

where *x* represents the input image, W_k represents the filter corresponding to the *k*th feature map, and *f* represents the specific activation function used[7]. Activation functions are utilized in order to increase the nonlinearity of the neural network, which prevents it from overfitting into linear and simple shapes.

The most popular activation function, **Rectified Linear Units layers (ReLU),** changes all negative pixel values within certain receptive fields into 0[8]. The resultant values of the receptive fields after RELU *f(x)* is thus computed by

$$f(x) = max\ (0, x)$$

where x is a pixel value of the input image[9].

After convolutional and ReLU layers, **pooling layers** are introduced to reduce the size of the previous feature maps and therefore lower the training time of the CNNs[10].

---

[7] Rawat, Waseem, and Zenghui Wang. “Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review.” *Neural Computation*, vol. 29, no. 9, 2017, pp. 2352–2449., doi:10.1162/neco_a_00990.

[8] Deshpande, Adit. “A Beginner's Guide To Understanding Convolutional Neural Networks Part 2.” *A Beginner's Guide To Understanding Convolutional Neural Networks*, Github, 29 July 2016, adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/.

[9] Li, Fei-Fei. “Convolutional Neural Networks for Image Recognition.” *CS231n Convolutional Neural Networks for Visual Recognition*, Stanford University, cs231n.github.io/neural-networks-1/.

[10] Deshpande, Adit. “A Beginner's Guide To Understanding Convolutional Neural Networks Part 2.” *A Beginner's Guide To Understanding Convolutional Neural Networks*, Github, 29 July 2016,

The most popular method is max pooling, which outputs the maximum pixel value within each subregion of the receptive fields. The values selected by RELU denoted by Y_{kij}, related to the *k*th feature map, satisfies the equation below:

$$Y_{kij} = \max_{\{p,q\} \in R_{ij}} x_{kpq}$$

where x_{kpq} refers to the pixel value located at (p,q) on the pooling region R_{ij}, and ij is the corresponding receptive field[11]. Relative locations are more significant than exact locations, so max pooling is generally desirable.

To avoid overfitting in CNNs, **dropout layers** drop out some values in the feature maps by setting them to zero[12]. This random action will prevent the neural networks from getting too fitted for the training data and allow them to perform with higher accuracies on the testing data.

Finally, **fully connected layers (FC)** take the outputs of the feature maps from the previous layers and use the softmax function to transform a N-dimensional vector

---

adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/.

[11] Rawat, Waseem, and Zenghui Wang. “Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review.” *Neural Computation*, vol. 29, no. 9, 2017, pp. 2352–2449., doi:10.1162/neco_a_00990.

[12] Deshpande, Adit. “A Beginner's Guide To Understanding Convolutional Neural Networks Part 2.” *A Beginner's Guide To Understanding Convolutional Neural Networks*, Github, 29 July 2016, adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/.

representative of the previous outputs into a one-dimensional vector[13]. This one-dimensional vector represents the probabilities that the input images are classified as cars or other categories, and the summation of its values should be one[14]. The transformation of the N-dimensional vector z into the one-dimensional vector $\sigma(z)$ can be modeled by the equation

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, 2, 3, \ldots \text{K.}$$ [15]

2.3 Hyperparameter Optimization

Although CNNs are generally effective at image classification, different neural networks may have distinct accuracies. The differences in their performances result from the different hyperparameters, which refer to the variable parameters in the architecture of the neural networks[16].

---

[13] Ujjwalkarn. “An Intuitive Explanation of Convolutional Neural Networks.” *An Intuitive Explanation of Convolutional Neural Networks*, The Data Science Blog, 11 Aug. 2016, ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/.

[14] Ujjwalkarn. “An Intuitive Explanation of Convolutional Neural Networks.” *An Intuitive Explanation of Convolutional Neural Networks*, The Data Science Blog, 11 Aug. 2016, ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/.

[15] Kulbear. “ReLU and Softmax Activation Functions.” *ReLU and Softmax Activation Functions*, Deep Learning Nano Foundation, 12 Feb. 2017, github.com/Kulbear/deep-learning-nano-foundation/wiki/ReLU-and-Softmax-Activation-Functions.

[16] Raffel, Colin. “Neural Network Hyperparameters.” *Neural Network Hyperparameters*, Colin Raffel, 17 Dec. 2015, colinraffel.com/wiki/neural_network_hyperparameters.

Hyperparameters include the number of convolutional layers, filter sizes, stride and padding values, dropout values, pooling methods, and many more[17]. For a given dataset of images, there exists a set of hyperparameters that yield the highest accuracy of predictions. Therefore, it is important to find the desired set of hyperparameters using certain optimization techniques.

This paper focuses on discussing the performance of Random Search and Tree-Structured Parzen Estimator (TPE) in optimizing hyperparameters for CNNs. Random Search involves selecting random subsets of parameters for optimization, while TPE chooses parameters and updates them iteratively to minimize error.

Random Search is selected for discussion in this paper, since it requires much less resources to train than its non-probabilistic counterpart, Grid Search. While Grid Search trains a model by testing all possible parameter combinations and selecting the best one, Random Search only tests random subsets of parameters, so it lowers the costs for training data[7].

In addition, TPE is chosen for this paper because it resolves the disadvantages of another popular probabilistic algorithm, Gaussian Processes[18]. Since the Gaussian Processes has many different configuration types, it can be difficult to select

---

[17] Pham, Vu. "Bayesian Optimization for Hyperparameter Tuning." *Bayesian Optimization for Hyperparameter Tuning*, Arimo, 18 Apr. 2016, arimo.com/data-science/2016/bayesian-optimization-hyperparameter-tuning/.

[18] Shevchuk, Yurii. "Hyperparameter Optimization for Neural Networks." *Hyperparameter Optimization for Neural Networks - NeuPy*, Neupy, 17 Dec. 2016, neupy.com/2016/12/17/hyperparameter_optimization_for_neural_networks.html#bayesian-optimization.

hyperparameters; however, TPE decides the hyperparameters at the end of each iteration based on the newly collected observations, so it is more efficient than Gaussian Processes[19].

2.4 Random Search Algorithm

The Random Search Algorithm is one of the most prevalent algorithms used in hyperparameter optimization for neural networks. To use random search algorithm, a set of hyperparameters and their ranges of values must first be defined[20].

After receiving the inputs of the hyperparameters and their possible values, Random Search randomly selects and runs inputted hyperparameter values[21]. In the figure below, the grid represents the hyperparameter search space and the blue points represent the hyperparameters chosen by Random Search. After each run, the accuracies and errors are recorded, so the optimal values for the hyperparameters can be achieved after comparing all results.

---

[19] Shevchuk, Yurii. “Hyperparameter Optimization for Neural Networks.” *Hyperparameter Optimization for Neural Networks - NeuPy*, Neupy, 17 Dec. 2016, neupy.com/2016/12/17/hyperparameter_optimization_for_neural_networks.html#bayesian-optimization.

[20] Shevchuk, Yurii. “Hyperparameter Optimization for Neural Networks.” *Hyperparameter Optimization for Neural Networks - NeuPy*, Neupy, 17 Dec. 2016, neupy.com/2016/12/17/hyperparameter_optimization_for_neural_networks.html#bayesian-optimization.

[21] Bergstra, James, and Yoshua Bengio. “Random Search for Hyper-Parameter Optimization.”*The Journal of Machine Learning Research*, JMLR.org, 1 Mar. 2012, dl.acm.org/citation.cfm?id=2503308.2188395.

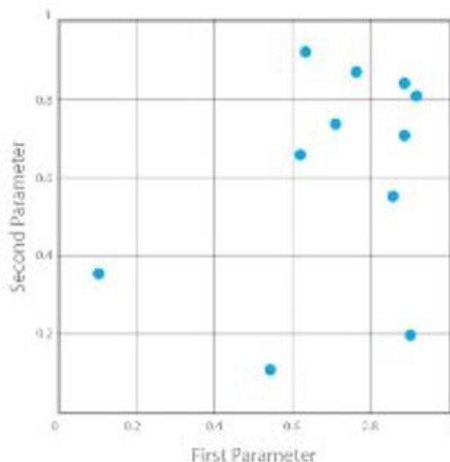

Figure 3 The Hyperparameter Search Space of Random Search[22]

By randomly selecting hyperparameter values, Random Search saves the processing power and execution time needed to try running all possible combinations of the hyperparameter values.

2.5 Tree-Structured Parzen Estimator Algorithm

Tree-Structured Parzen Estimator (TPE) is an efficient hyperparameter optimization algorithm that performs many iterations to train data. During each iteration, it collects new observations on the corresponding accuracies of the hyperparameters[23]. At the end of each iteration, the algorithm decides on the hyperparameters that should be

[22] "Contest 2nd Place: Automating Data Science." *Automating Data Science*, KdNuggets, www.kdnuggets.com/2016/08/automating-data-science.html.

[23] Bergstra, James S., et al. "NIPS Proceedingsβ." *Algorithms for Hyper-Parameter Optimization*, 12 Dec. 2011, papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.

tried next. In a general sense, it stores information of past iterations and uses the previous results for the following iterations.

For the first iteration of TPE, it must collect some data to work as past iterations before it can start optimizing the hyperparameters[24]. To collect data, TPE can use any of the existing hyperparameter techniques, such as Grid Search, Random Search, or Hand Tuning.

After TPE has collected sufficient data, TPE separates them into two groups. The first group contains all the hyperparameters that yield good results after evaluation, while the second group contains the rest[25]. For the following iterations, the TPE aims to discover the set of hyperparameters that have higher probability to fall in the first group than the second group[26]. To achieve this goal, it attempts to maximize the Expected Improvement for each hyperparameter, which can be written as

---

[24] Shevchuk, Yurii. “Hyperparameter Optimization for Neural Networks.” *Hyperparameter Optimization for Neural Networks - NeuPy*, Neupy, 17 Dec. 2016, neupy.com/2016/12/17/hyperparameter_optimization_for_neural_networks.html#bayesian-optimization.

[25] Shevchuk, Yurii. “Hyperparameter Optimization for Neural Networks.” *Hyperparameter Optimization for Neural Networks - NeuPy*, Neupy, 17 Dec. 2016, neupy.com/2016/12/17/hyperparameter_optimization_for_neural_networks.html#bayesian-optimization.

[26] Shevchuk, Yurii. “Hyperparameter Optimization for Neural Networks.” *Hyperparameter Optimization for Neural Networks - NeuPy*, Neupy, 17 Dec. 2016, neupy.com/2016/12/17/hyperparameter_optimization_for_neural_networks.html#bayesian-optimization.

$$EI(x) = \frac{f(x)}{s(x)}$$

where $f(x)$ represents the probability that the selected data falls in the first group while $s(x)$ is the probability that it falls in the second group[27].

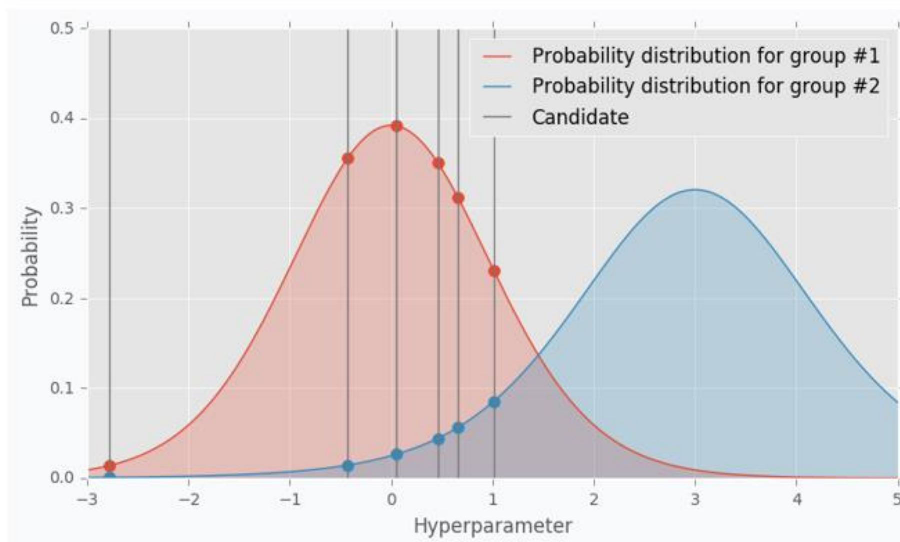

Figure 4 Probability distributions of Group 1 and 2 for a few Hyperparameters

28

[27] Shevchuk, Yurii. "Hyperparameter Optimization for Neural Networks." *Hyperparameter Optimization for Neural Networks - NeuPy*, Neupy, 17 Dec. 2016, neupy.com/2016/12/17/hyperparameter_optimization_for_neural_networks.html#bayesian-optimization.

[28] Shevchuk, Yurii. "Hyperparameter Optimization for Neural Networks." *Hyperparameter Optimization for Neural Networks - NeuPy*, Neupy, 17 Dec. 2016, neupy.com/2016/12/17/hyperparameter_optimization_for_neural_networks.html#bayesian-optimization.

3. Methodology to Compare Random Search and Tree-Structured Parzen Estimator

3.1 Experiments Design

Since there are infinite CNN architectures due to the vast quantity of hyperparameters, this paper focuses on only two custom CNNs for simplicity: simple CNN and complex CNN. The simple CNN has relatively less layers and hyperparameters to optimize, while the complex CNN has relatively more to optimize. Details to the architectures of simple CNN and complex CNN will be provided in the later sections (Section 4.1, 5.1).

For the experiments, a simple CNN and a complex CNN are built to train and test images from an image dataset. For every chosen hyperparameter, Random Search and TPE are run for each CNN, and the results are compared. For each hyperparameter case, all other hyperparameter values are predefined since they are control variables.

The efficiencies of the two algorithms for each hyperparameter case are determined by the accuracies and errors of their optimized CNNs. Higher accuracies and lower errors of the optimized CNNs indicate that the hyperparameter algorithm produces better performance.

The execution times of the runs are also recorded to compare the time both algorithms use because time is usually an important factor for consideration, especially for trainings of large CNNs.

The testing environment is a personal laptop with processor i7-4720HQ that clocks up to 2.6 GHz, Nvidia GTX950M, and 12 GB RAM.

3.2 Chosen Hyperparameters

For both simple and complex CNNs, the following hyperparameters are selected for testing:

- **The number of convolutional layers**: the number of convolutional layers determines how many convolutions the filters perform across the images and thus heavily affects the classifying accuracy of the CNN.
- **Batch size**: batch size, which refers to the number of training examples utilized in one iteration, determines the number of times the neurons' weights are updated and therefore affects the performance of the CNN[29].
- **Activation functions**: different activation functions introduce different forms of non-linearity into the CNN, which influences its variability and accuracy.
- **Dropout values**: the dropout values influence the CNN's ability to respond and correct the overfitting that happens within the network.
- **Optimizer method**: the optimizer methods minimize the loss functions generated iteratively at the output layer and determine the optimized weights for each neuron, affecting the performance of the network.

[29] Gaillard, Frank. "Batch Size (Machine Learning)." *Batch Size (Machine Learning)*, Radiopaedia, radiopaedia.org/articles/batch-size-machine-learning.

3.3 Libraries Descriptions

All code is written in Python, as Python provides many convenient, state-of-the-art machine learning libraries for use.

The library used for building the structure of the CNN is Keras. Keras is a high-level neural network API that is written in Python and capable of using popular machine learning libraries for backend processing tasks[30]. The experiments use the TensorFlow library, which is an open source software for numerical computation, for providing the methods needed to train and test data[31].

Furthermore, the Hyperas library is used to provide the implementation for Random Search and TPE for Keras models[32].

[30] "Keras: The Python Deep Learning Library." *Keras Documentation*, Keras, keras.io/.

[31] "TensorFlow." *TensorFlow*, Tensorflow, www.tensorflow.org/.

[32] "Hyperas." *Hyperas by Maxpumperla*, Github, maxpumperla.github.io/hyperas/.

3.4 Dataset Descriptions

For the image dataset, the popular CIFAR-10 dataset is used for the entire training, testing, and optimizing tasks. Details of the CIFAR-10 dataset are provided below[33].

CIFAR-10 dataset	
Number of Images	60000
Number of Classes	10
Images Per Class	6000
Number of Training Images	50000
Number of Testing Images	10000
Pixels of Each Image	32x32
Classes	Airplane
	Automobile
	Bird
	Cat
	Deer
	Dog
	Frog
	Horse
	Ship
	Truck

Table 1 CIFAR-10 Dataset Information

[33] Krizhevsky, Alex. "The CIFAR-10 Dataset." *CIFAR-10 and CIFAR-100 Datasets*, University of Toronto, www.cs.toronto.edu/~kriz/cifar.html.

4. Comparing Random Search and Tree-Structured Parzen Estimator Algorithms for Simple CNN

4.1 Architecture of Simple CNN

The simple CNN used in this paper has the following architecture: 4 convolutional layers, 6 activation layers, 2 max-pooling layers, and 3 dropout layers. The first convolutional layer acts as the input layer, while the last activation layer acts as the output layer.

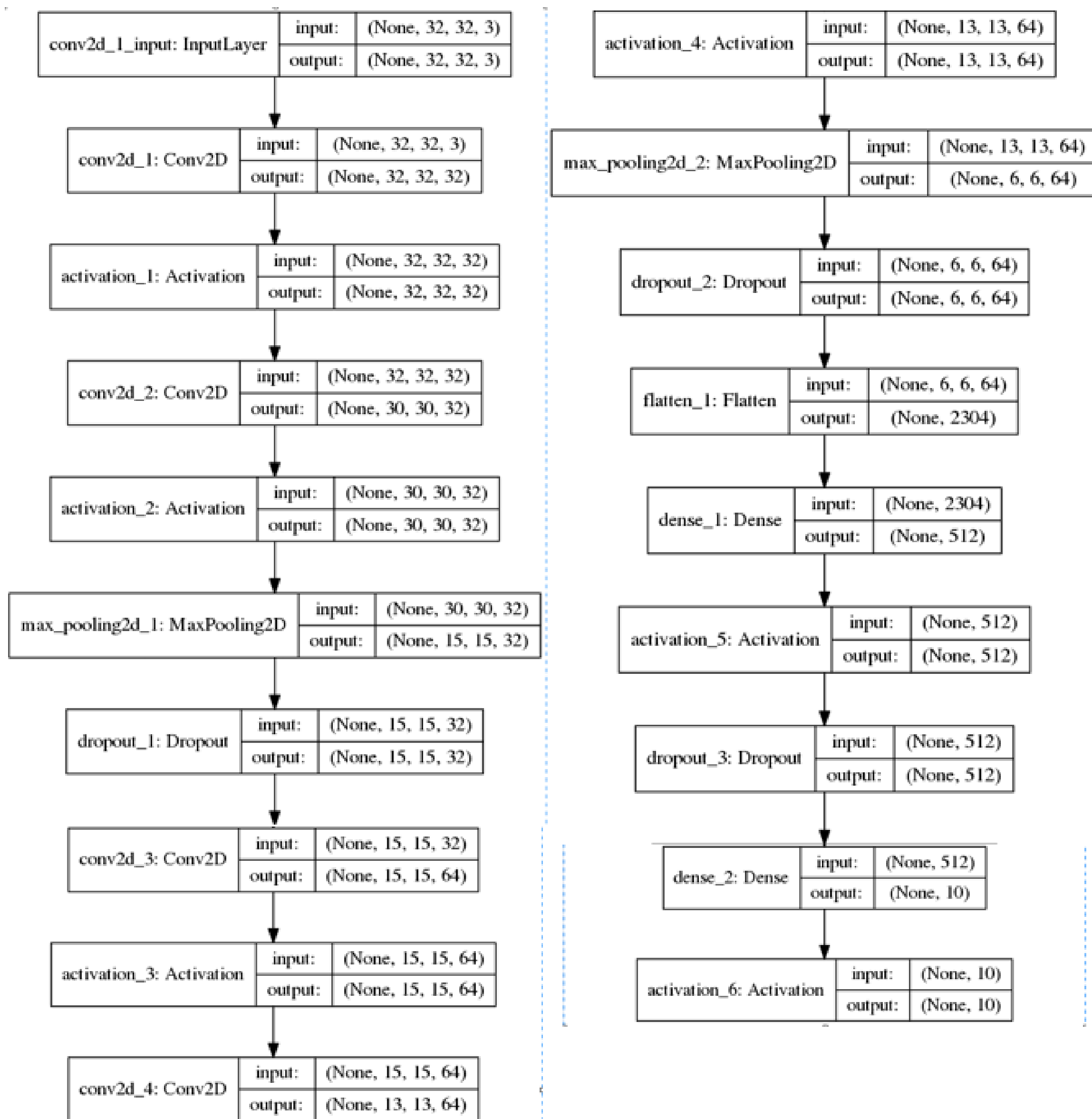

Figure 5 Architecture of the Simple CNN

4.2 Optimal Number of Convolutional Layers

The code tests the optimal number of convolutional layers in the simple CNN (See Appendix A.1). The number of convolutional layers can be either 2 or 4.

	Accuracy	Loss	Average Time Per Epoch (s)	Total Time (hr)
Random Search	0.72970	0.81089	197	4:56
TPE	0.71450	0.82815	199	4:59

Table 2 Accuracy, Loss, and Execution Times for Number of Convolutional Layers

Hyperparameter	Value
Number of Convolutional Layers	4

Table 3 Optimal Number of Convolutional Layers

The tables show that Random Search’s accuracy 0.72970 is higher than TPE’s accuracy 0.71450, while Random Search’s loss 0.81089 is lower than TPE’s loss 0.82815, both of which indicate Random Search’s better performance for optimizing the number of convolutional layers in the simple CNN.

Although the difference in execution times, 3 minutes, is small, Random Search still runs a little faster than TPE. Therefore, combining all accuracy, loss, and time factors, Random Search has better performance in this case.

4.3 Optimal Batch Size

The code tests the optimal batch size in the simple CNN. The batch size is selected from the values: 16, 32, 64, 128, 256, and 512 (See Appendix A.2).

Algorithm	Accuracy	Loss	Average Time Per Epoch (s)	Total Time (hr)
Random Search	0.62680	1.08006	195	4:53
TPE	0.64380	1.10891	215	5:23

Table 4 Accuracy, Loss, and Execution Times for Batch Size

Hyperparameter	Value
Batch Size	64

Table 5 Optimal Batch Size

The tables show that TPE has higher accuracy than Random Search, 0.64380 compared with 0.62680, respectively. However, the loss of Random Search is 1.08006, smaller than the loss of TPE 1.10891. From the data, no conclusion about the efficiency of two algorithms can be made.

Still, Random Search uses significantly less time to optimize the batch size than TPE, as Random Search takes 4 hours and 53 minutes while TPE takes 5 hours and 23 minutes.

4.4 Optimal Activation Functions

The code tests the simple CNN for the optimal activation functions (See Appendix A.3). The optimized activation functions can be "relu," "sigmoid," or "tanh."

Algorithm	Accuracy	Loss	Average Time Per Epoch (s)	Total Time (hr)
Random Search	0.10000	2.30259	195	4:53
TPE	0.10000	2.30259	207	5:11

Table 6 Accuracy, Loss, and Execution Times for Activation Functions

Hyperparameter	Function
Activation Layer 1	sigmoid
Activation Layer 2	sigmoid
Activation Layer 3	sigmoid
Activation Layer 4	tanh
Activation Layer 5	tanh
Activation Layer 6	tanh
Activation Layer 7	sigmoid
Activation Layer 8	sigmoid
Activation Layer 9	sigmoid

Table 7 Optimal Activation Functions

As the tables indicate, the accuracy and loss of the model optimized using Random Search and TPE are both identical. This shows that they are both very inefficient at optimizing activation functions considering the extremely low accuracies and high losses.

However, Random Search still computes faster than TPE, 4 hours and 53 minutes compared with 5 hours and 11 minutes.

4.5 Optimal Dropout Values

The code tests optimal parameters of the dropout values in the simple CNN (See Appendix A.4). The optimized dropout values can be any number between 0 and 1, exclusive.

Algorithm	Accuracy	Loss	Average Time Per Epoch (s)	Total Time (hr)
Random Search	0.66720	0.98590	207	5:11
TPE	0.65580	1.02881	210	5:15

Table 8 Accuracy, Loss, and Execution Times for Dropout Values

Hyperparameter	Value
Dropout Layer 1	0.112460
Dropout Layer 2	0.320753
Dropout Layer 3	0.734215
Dropout Layer 4	0.692539
Dropout Layer 5	0.212800

Table 9 Optimal Dropout Values

As the results show, Random Search is more efficient than TPE when they optimize dropout values in the simple CNN. Random Search has an accuracy of 0.66720, higher than TPE's 0.65580, and the loss of Random Search 0.98590 is also smaller than TPE's 1.02881.

The time spent on running hyperparameter optimization on dropout values is about the same, although Random Search takes slightly less time than TPE, 5 hours and 11 minutes compared with 5 hours and 15 minutes.

4.6 Optimal Optimizer Method

The possible values for the optimizer method are “rmsprop,” “adam,” “sgd,” “adagrad,” “adadelta,” “adamax,” and “nadam.”[34] The code tests for the optimal optimizer method within the simple CNN (See Appendix A.5).

	Accuracy	Loss	Average Time Per Epoch (s)	Total Time
Random Search	0.798800	0.582790	210	5:15
TPE	0.7968	0.583918	208	5:12

Table 10 Accuracy, Loss, and Execution Times for Optimizer Method

	Method
Optimizer Method	adamax

Table 11 Optimal Optimizer Method

The performance of Random Search and TPE to optimize the optimizer method is similar: their accuracies are the same by the hundredth, with Random Search’s accuracy 0.79880 slightly higher than TPE’s 0.79680. Random Search’s loss is also lower, indicating Random Search’s better performance.

With higher accuracy, Random Search also runs with less execution time, so Random Search is more efficient than TPE for optimizing the optimizer method.

[34] “Optimizers.” *Optimizers - Keras Documentation*, Keras, keras.io/optimizers/.

4.7 Optimal Values for All Chosen Hyperparameters

After evaluating the efficiency of the two algorithms for different hyperparameters, it is significant to see their performances when all hyperparameters are optimized simultaneously. This is generally useful for most hyperparameter testing cases that attempt to optimize a neural network to yield the best accuracy by optimizing every possible hyperparameters.

The code tests for all chosen hyperparameters within the simple CNN (See Appendix A.6).

	Accuracy	Loss	Average Time Per Epoch (s)	Total Time (hr)
Random Search	0.49950	1.39453	208	5:12
TPE	0.33940	1.65564	220	5:30

Table 12 Accuracy, Loss, and Execution Times for All Previous Hyperparameters

Hyperparameter	Value/Function
Number of Convolutional Layers	4
Batch Size	256
Activation Layer 1	relu
Activation Layer 2	tanh
Activation Layer 3	sigmoid
Activation Layer 4	sigmoid
Activation Layer 5	relu
Activation Layer 6	tanh
Activation Layer 7	tanh
Activation Layer 8	sigmoid
Dropout Layer 1	0.711425
Dropout Layer 2	0.562382
Dropout Layer 3	0.394994
Dropout Layer 4	0.975819
Optimizer Method	rmsprop

Table 13 Optimal Values for All Previous Hyperparameters

The tables clearly demonstrate that Random Search produces better performance than TPE, since Random Search's accuracy 0.49950 is higher than TPE's accuracy 0.33940, and Random Search's loss 1.39453 is lower than 1.65564.

Similar to other previous hyperparameter cases, Random Search also has time advantage because it takes relatively less time to train using its algorithm than TPE.

5. Comparing Random Search and Tree-Structured Parzen Estimator for Complex CNN

5.1 Architecture of Complex CNN

The complex CNN used in this paper has the following architecture: 8 convolutional layers, 12 activation layers, 2 max pooling layers, and 6 dropout layers. The first convolutional layer acts as the input layer, while the last activation layer acts as the output layer.

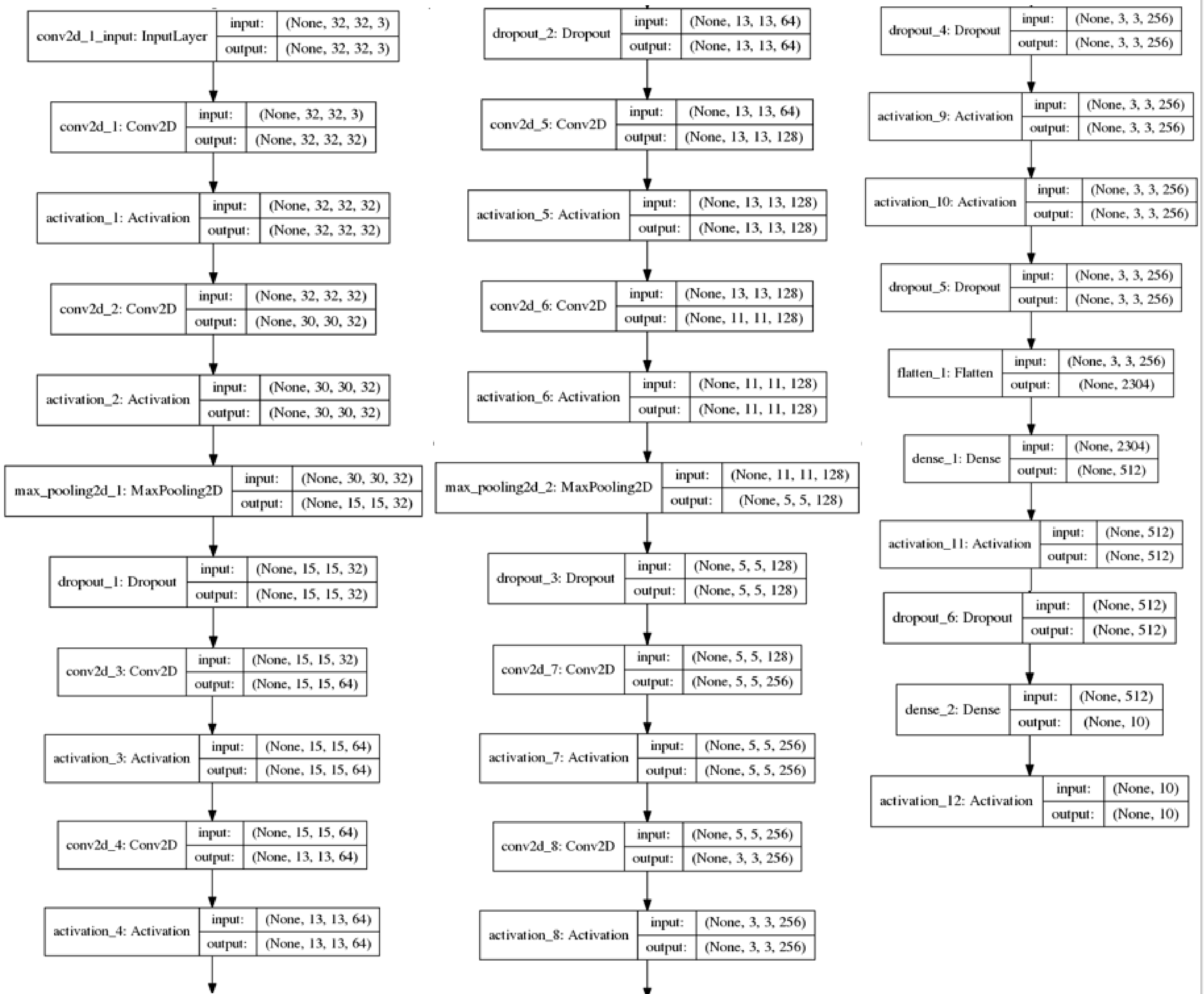

Figure 6 Architecture of the Complex CNN

5.2 Optimal Number of Convolutional Layers

The code tests for all hyperparameters for the complex CNN (See Appendix B.1). The possible optimal number of convolutional layers can be either 6 or 8.

	Accuracy	Loss	Average Time Per Epoch (s)	Total Time
Random Search	0.62450	1.30024	369	9:14
TPE	0.77280	0.92918	385	9:38

Table 14 Accuracy, Loss, and Execution Times for the Number of Convolutional Layers

Hyperparameter	Value
Number of Convolutional Layers	20

Table 15 Optimal Number of Convolutional Layers

The tables show that TPE has higher accuracy 0.77280 and lower loss 0.92918 than Random Search's accuracy 0.62450 and loss 1.30024. In contrast with the outcomes presented in the simple CNN, TPE offers better performance for optimizing the number of convolutional layers in the complex CNN.

In terms of execution times, Random Search runs faster than TPE. This is consistent with the simple CNN for testing the number of convolutional layers.

However, the difference in execution times between two algorithms in two architectures are very different. In the simple architecture, there is only a 3-minute difference, but for the complex architecture, there is a 24-minute difference (See Section 4.2). This means that higher complexity of CNN may produce more significant difference

in execution times and therefore also induce considerations of time as an important factor in deciding which algorithm to use.

5.3 Optimal Batch Size

The code tests for the optimal batch size within the complex CNN (See Appendix B.2).

	Accuracy	Loss	Average Time Per Epoch (s)	Total Time
Random Search	0.64430	0.59941	329	8:13
TPE	0.72100	0.75302	323	8:25

Table 16 Accuracy, Loss, and Execution Times for Batch Size

Hyperparameter	Value
Batch Size	128

Table 17 Optimal Batch Size

Similar to those from the simple CNN, the results indicate that it is unclear which algorithm performs better for optimizing batch size in complex CNN. TPE has higher accuracy yet also higher loss than Random Search.

For execution time, Random Search, in this case, runs 12 minutes faster than TPE.

5.4 Optimal Activation Functions

The code tests for the optimal activation functions within the complex CNN (See Appendix B.3).

	Accuracy	Loss	Average Time Per Epoch (s)	Total Time
Random Search	0.10000	2.30259	349	8:53
TPE	0.10000	2.30259	360	9:00

Table 18 Accuracy, Loss, and Execution Times for Activation Functions

Hyperparameter	Function
Activation Layer 1	sigmoid
Activation Layer 2	sigmoid
Activation Layer 3	tanh
Activation Layer 4	tanh
Activation Layer 5	sigmoid
Activation Layer 6	tanh
Activation Layer 7	tanh
Activation Layer 8	relu
Activation Layer 9	sigmoid
Activation Layer 10	sigmoid
Activation Layer 11	relu
Activation Layer 12	sigmoid
Activation Layer 13	sigmoid
Activation Layer 14	tanh
Activation Layer 15	tanh
Activation Layer 16	relu
Activation Layer 17	relu
Activation Layer 18	sigmoid
Activation Layer 19	sigmoid
Activation Layer 20	relu

Table 19 Optimal Activation Functions

Interestingly, the results of accuracy and loss are exactly the same for both simple and complex CNNs. Random Search and TPE score on the same level based on accuracy and loss.

However, Random Search, again, uses less time to run than TPE, making TPE having less advantage in execution times.

5.5 Optimal Dropout Values

The code tests for the optimal dropout values within the complex CNN (See Appendix B.4).

	Accuracy	Loss	Average Time Per Epoch (s)	Total Time
Random Search	0.46314	1.30264	367	9:10
TPE	0.56426	1.20152	378	9:27

Table 20 Accuracy, Loss, and Execution Times for Dropout Values

Hyperparameter	Value
Dropout Layer 1	0.112460
Dropout Layer 2	0.320753
Dropout Layer 3	0.734215
Dropout Layer 4	0.692539
Dropout Layer 5	0.302848
Dropout Layer 6	0.832430
Dropout Layer 7	0.386926
Dropout Layer 8	0.206659
Dropout Layer 9	0.532913
Dropout Layer 10	0.492853
Dropout Layer 11	0.513511
Dropout Layer 12	0.734525

Table 21 Optimal Dropout Values

For optimizing dropout values in the complex CNN, TPE has higher accuracy 0.56426 than Random Search 0.46314. It also performs with less loss 1.20152 than Random Search 1.30264. This clearly points out that TPE yields better performance.

However, like the rest of the cases, TPE still runs 17 minutes slower than Random Search and may grow if an even more complex CNN is trained.

5.6 Optimal Optimizer Method

The code tests for the optimal optimizer method within the complex CNN (See Appendix B.5).

	Accuracy	Loss	Average Time Per Epoch (s)	Total Time
Random Search	0.50000	1.30264	357	8:56
TPE	0.50000	1.30264	371	9:17

Table 22 Accuracy, Loss, and Execution Times for Optimizer Method

Hyperparameter	Method
Optimizer Method	adam

Table 23 Optimal Optimizer Method

The tables demonstrate that Random Search and TPE both have the same results for accuracy and loss. Thus, they have equal performance for optimizing optimal function in the complex CNN.

For execution times, TPE runs 21 minutes slower than Random Search, which needs to be taken into consideration when applying either algorithm.

5.7 Optimal Values for All Chosen Hyperparameters

The code tests for all hyperparameters within the complex CNN (See Appendix B.6).

	Accuracy	Loss	Average Time Per Epoch (s)	Total Time
Random Search	0.33573	1.90258	361	9:02
TPE	0.41583	1.86930	399	9:59

Table 24 Accuracy, Loss, and Execution Times for All Previous Hyperparameters

Hyperparameter	Value	Hyperparameter	Value
Number of Convolutional Layers	20	Activation Layer 17	relu
Batch Size	128	Activation Layer 18	tanh
Activation Layer 1	relu	Activation Layer 19	sigmoid
Activation Layer 2	sigmoid	Activation Layer 20	relu
Activation Layer 3	relu	Dropout Layer 1	0.115360
Activation Layer 4	tanh	Dropout Layer 2	0.36722
Activation Layer 5	tanh	Dropout Layer 3	0.47274
Activation Layer 6	relu	Dropout Layer 4	0.78302
Activation Layer 7	sigmoid	Dropout Layer 5	0.314690
Activation Layer 8	relu	Dropout Layer 6	0.205615
Activation Layer 9	sigmoid	Dropout Layer 7	0.32651
Activation Layer 10	sigmoid	Dropout Layer 8	0.20604
Activation Layer 11	sigmoid	Dropout Layer 9	0.50159
Activation Layer 12	tanh	Dropout Layer 10	0.63852
Activation Layer 13	tanh	Dropout Layer 11	0.51366
Activation Layer 14	relu	Dropout Layer 12	0.73569
Activation Layer 15	sigmoid	Optimizer Method	adam
Activation Layer 16	relu		

Table 25 Optimal Values for all Previous Hyperparameters

For the optimization of all chosen hyperparameters, TPE is explicitly shown to perform better in the complex CNN. It performs with better accuracy 0.41583 than Random Search 0.33573 and with less loss 1.86930 than Random Search 1.90258.

However, Random Search uses much less time than TPE in a complex CNN. For complex CNN, there is a massive 57 minutes time difference between the execution times of the two algorithms. For more intensive trainings that would take up days or weeks, the time difference could scale up and be the deciding factor to use Random Search even though TPE tends to yield higher accuracy.

6. Comparing the Efficiency of Random Search and Tree-Structured Parzen Estimator for All Chosen Hyperparameters in Simple and Complex CNNs

6.1 Accuracy

The graphs below are all plotted with accuracy against the number of epochs during hyperparameter optimization for all the chosen hyperparameters.

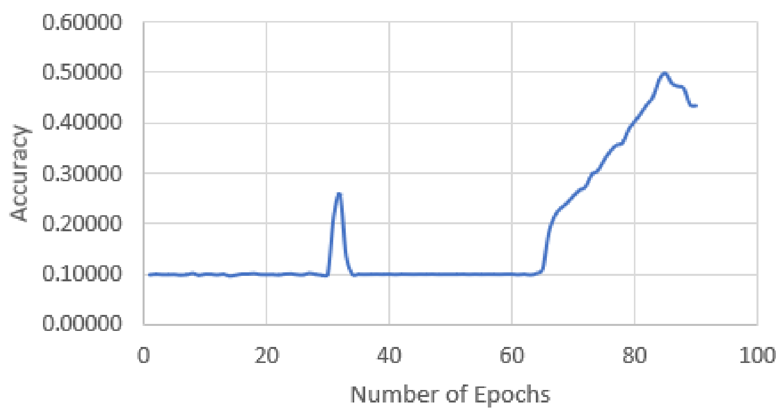

Figure 7 Graph of Accuracy Against Number of Epochs of Random Search for Simple CNN

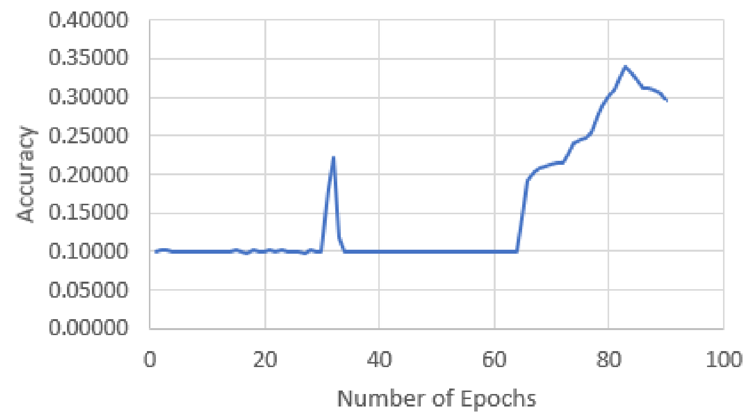

Figure 8 Accuracy Against Number of Epochs of Random Search for Complex CNN

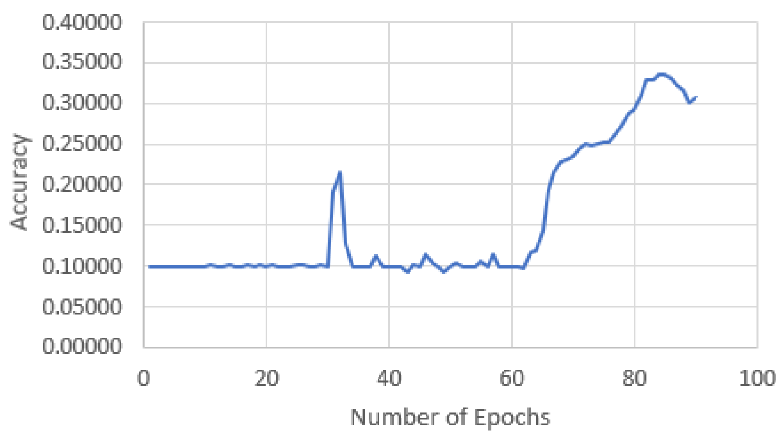

Figure 9 Graph of Accuracy Against Number of Epochs of TPE for Simple CNN

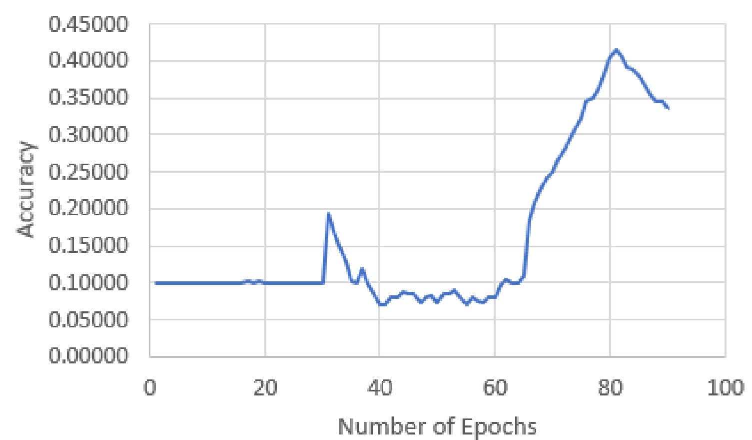

Figure 10 Graph of Accuracy Against Number of Epochs of TPE for Complex CNN

Figure 7 and *Figure 8,* which both show the performance of Random Search, demonstrate a difference in the accuracy. Although the shapes of the graphs are similar, Random Search scores higher on accuracy in the simple CNN than in the complex CNN.

However, *Figure 9* and *Figure 10* indicate that TPE has better accuracy in complex CNN than simple CNN, as evidenced by the former's higher data points in the later epochs.

6.2 Loss

The graphs below are all plotted with loss against the number of epochs during hyperparameter optimization for all the chosen hyperparameters.

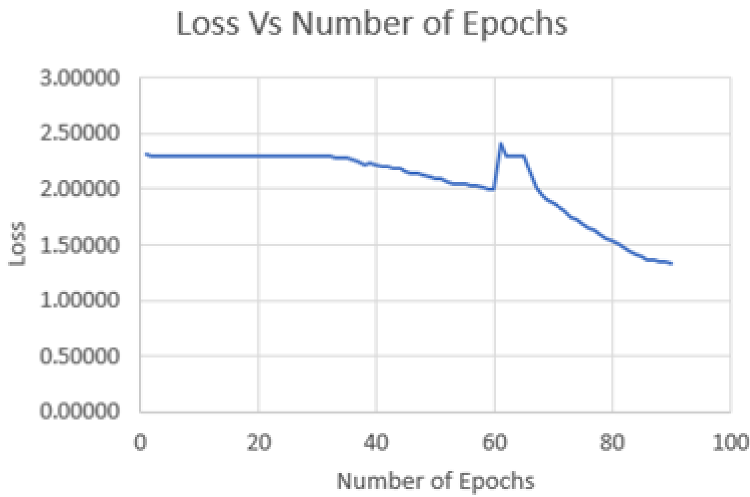

Figure 11 Graph of Loss Against Number of Epochs of Random Search for Simple CNN

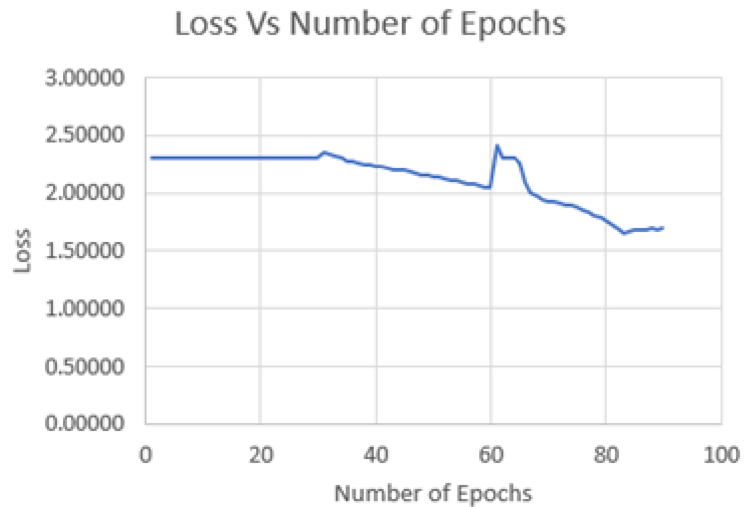

Figure 12 Graph of Loss Against Number of Epochs of Random Search for Complex CNN

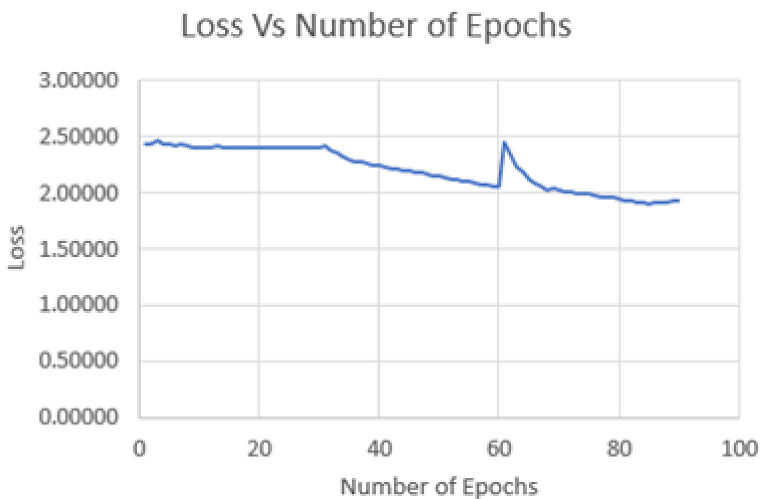

Figure 13 Graph of Loss Against Number of Epochs of TPE for Simple CNN

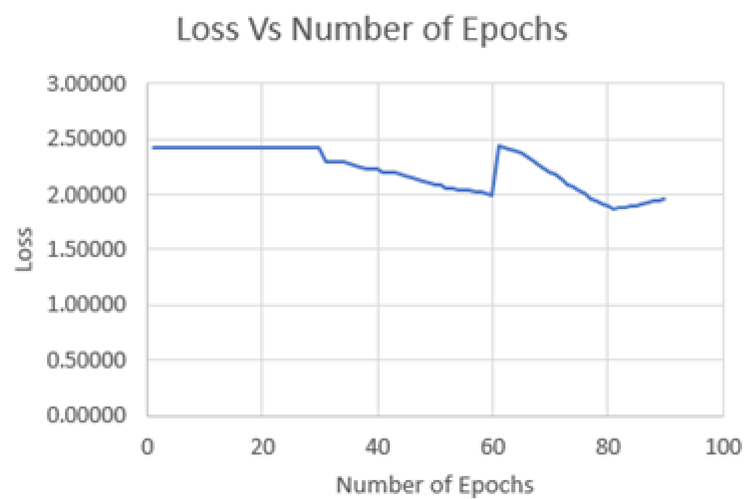

Figure 14 Graph of Loss Against Number of Epochs of TPE for Complex CNN

By comparing the graphs in *Figure 11* and *Figure 12*, it can be deduced that Random Search shows less loss in simple CNN, since its data points are all lower than that of TPE.

Yet, *Figure 13* and *Figure 14* addresses TPE's lower loss in complex CNN than in simple CNN, since the latter figure has lower points for the later epochs.

7. Conclusion

The tables below summarize the results of the experiments conducted throughout the entire hyperparameter optimization process for both simple and complex CNNs.

	Accuracy	Loss	Time	Accuracy	Loss	Time	
Number of Convolutional Layers	0.72970	0.81089	4:56	0.71450	0.82815	4:59	Random Search
Batch Size	0.62680	1.08006	4:53	0.64380	1.10891	5:23	Inconclusive
Activation Functions	0.10000	2.30259	4:53	0.10000	2.30259	5:11	Same
Dropout Values	0.66720	0.98590	5:11	0.65580	1.02881	5:15	Random Search
Optimizer Method	0.798800	0.582790	5:15	0.7968	0.583918	5:12	Random Search
All	0.49950	1.39453	5:12	0.33940	1.65564	5:30	Random Search

Table 26 Summary of Accuracy, Loss, Execution Times, and the More Efficient Algorithm for All Hyperparameters in Simple CNN

	Random Search			TPE			More Efficient Algorithm
Hyperparameter	Accuracy	Loss	Time	Accuracy	Loss	Time	
Number of Convolutional Layers	0.6245	1.30024	9:14	0.7728	0.92918	9:38	TPE
Batch Size	0.64430	0.59941	8:13	0.72100	0.75302	8:25	Inconclusive
Activation Functions	0.10000	2.30259	8:53	0.10000	2.30259	9:00	Same
Dropout Values	0.46314	1.30264	9:10	0.56426	1.20152	9:27	TPE
Optimizer Method	0.50000	1.30264	8:56	0.50000	1.30264	9:17	Same
All	0.33573	1.90258	9:48	0.41583	1.86930	9:59	TPE

Table 27 Summary of Accuracy, Loss, Execution Times, and the More Efficient Algorithm for All Hyperparameters in Complex CNN

In simple CNNs, when optimizing the number of convolutional layers, the dropout values, optimizer method, and all chosen hyperparameters, Random Search is more efficient. However, for activation functions, both algorithms give the same performance in terms of accuracy and loss, so they have equal efficiency. Lastly, the efficiency of the algorithms for batch size is inconclusive because Random Search has lower accuracy and lower loss than TPE.

For complex CNNs, TPE is more efficient for optimizing the number of convolutional layers, dropout values, and all chosen hyperparameters. However, in optimizing activation functions and the optimizer method, TPE and Random Search perform equally. There is also no conclusion for batch size again, since TPE has higher accuracy and higher loss than Random Search.

Based on accuracy and loss, Random Search can be generally considered as more efficient for optimizing simple CNNs, while TPE is more efficient for complex CNNs.

For execution time, TPE is found to take more time than Random Search in both types of architectures, so Random Search has the advantage in time. For simple CNNs, it is reasonable to use Random Search since it not only produces higher accuracy and lower loss but also consumes less time for most cases. However, for complex CNNs, TPE yields more accurate results, but Random Search takes up significantly less time. Thus, the purpose of the model, available resources, and difference in execution times between two algorithms, as well as other relevant factors, all need to be considered before deciding to choose Random Search or TPE.

8. Limitations of this Paper

There are several limitations to this paper. First, this paper only discusses the basic hyperparameters for CNNs: the number of convolutional layers, batch size, activation function, dropout value, and the optimizer method. The hyperparameters are not limited to these; in fact, there are countless hyperparameters within any neural network.

Due to the processing power limitations and time restraints, the CNNs cannot be trained for larger number of epochs. Generally, the longer the training time is, the better the performance of an optimized CNN.

In addition, the architecture for the CNNs are customarily built primarily for simplicity and convenience, and thus may not yield the best possible network architecture for the purpose of image recognition.

Furthermore, this paper only discusses hyperparameter optimization for image recognition which uses the CIFAR-10 image dataset. Although the dataset is one of the most comprehensive image dataset available on the Internet today, it only consists of a small fraction of familiar images, and hence, could not be generalized for the recognition of other images.

9. Further Investigation

To further investigate this topic, more hyperparameters should be tested to evaluate the performances of Random Search and TPE algorithms. Selection of more hyperparameters would allow more computer scientists and researchers to acknowledge the algorithms' performance on a wide variety of hyperparameters and will alleviate their time and resources to discover them themselves.

Future explorations around this topic can also focus on the efficiency of Random Search and TPE in other types of artificial neural networks, such as feed forward networks or recurrent neural networks. Although CNN is the most popular neural network used now, pursuing this research would be helpful to enhance the performances of other neural networks.

10. Bibliography

Bergstra, James, and Yoshua Bengio. “Random Search for Hyper-Parameter Optimization.”*The Journal of Machine Learning Research*, JMLR.org, 1 Mar. 2012, dl.acm.org/citation.cfm?id=2503308.2188395.

Bergstra, James S., et al. “NIPS Proceedingsβ.” *Algorithms for Hyper-Parameter Optimization*, 12 Dec. 2011, papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.

“Contest 2nd Place: Automating Data Science.” *Automating Data Science*, KdNuggets, www.kdnuggets.com/2016/08/automating-data-science.html.

“Convolutional Neural Network.” *Convolutional Neural Network*, Matlab, www.mathworks.com/discovery/convolutional-neural-network.html.

Deshpande, Adit. “A Beginner's Guide To Understanding Convolutional Neural Networks Part 2.” *A Beginner's Guide To Understanding Convolutional Neural Networks*, Github, 29 July 2016, adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/.

Gaillard, Frank. “Batch Size (Machine Learning).” *Batch Size (Machine Learning)*, Radiopaedia, radiopaedia.org/articles/batch-size-machine-learning.

“Hyperas.” *Hyperas by Maxpumperla*, Github, maxpumperla.github.io/hyperas/.

“Keras: The Python Deep Learning Library.” *Keras Documentation*, Keras, keras.io/.

Krizhevsky, Alex. “The CIFAR-10 Dataset.” *CIFAR-10 and CIFAR-100 Datasets*, University of Toronto, www.cs.toronto.edu/~kriz/cifar.html.

Kulbear. “ReLU and Softmax Activation Functions.” *ReLU and Softmax Activation Functions*, Deep Learning Nano Foundation, 12 Feb. 2017, github.com/Kulbear/deep-learning-nano-foundation/wiki/ReLU-and-Softmax-Activation-Functions.

Li, Fei-Fei. “Image Classification.” *CS231n Convolutional Neural Networks for Visual Recognition*, Stanford University, cs231n.github.io/classification/.

Li, Fei-Fei. “Convolutional Neural Networks for Image Recognition.” *CS231n Convolutional Neural Networks for Visual Recognition*, Stanford University, cs231n.github.io/neural-networks-1/.

Pham, Vu. “Bayesian Optimization for Hyperparameter Tuning.” *Bayesian Optimization for Hyperparameter Tuning*, Arimo, 18 Apr. 2016, arimo.com/data-science/2016/bayesian-optimization-hyperparameter-tuning/.

Raffel, Colin. “Neural Network Hyperparameters.” *Neural Network Hyperparameters*, Colin Raffel, 17 Dec. 2015, colinraffel.com/wiki/neural_network_hyperparameters.

Rawat, Waseem, and Zenghui Wang. "Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review." *Neural Computation*, vol. 29, no. 9, 2017, pp. 2352–2449., doi:10.1162/neco_a_00990.

Russel, Bryan. "Object Recognition by Scene Alignment." *Object Recognition by Scene Alignment*, Bryan Russel, bryanrussell.org/projects/recognitionBySceneAlignment/index.html.

Shevchuk, Yurii. "Hyperparameter Optimization for Neural Networks." *Hyperparameter Optimization for Neural Networks - NeuPy*, Neupy, 17 Dec. 2016, neupy.com/2016/12/17/hyperparameter_optimization_for_neural_networks.html#bayesian-optimization.

"TensorFlow." *TensorFlow*, Tensorflow, www.tensorflow.org/.

Ujjwalkarn. "An Intuitive Explanation of Convolutional Neural Networks." *An Intuitive Explanation of Convolutional Neural Networks*, The Data Science Blog, 11 Aug. 2016, ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/.

11. Appendix

A. Source Code of Hyperparameter Optimization for Simple CNN

A.1 Optimal Number of Convolutional Layers

```
1.  def model(datagen, X_train, Y_train, X_test, Y_test):
2.
3.      # number of epochs
4.      nb_epoch = 30
5.
6.      # batch size
7.      batch_size = 64
8.
9.      # input image dimensions
10.     img_rows, img_cols = 32, 32
11.
12.     # image channels
13.     img_channels = 3
14.
15.     model = Sequential()
16.
17.     # Convolution2D: Convolutional Layer
18.     # Activation: Activation Layer
19.     # MaxPooling2D: MaxPooling Layer
20.     # Dropout: Dropout Layer
21.     # optimizer: Optimizer Method
22.
23.     model = Sequential()
24.
25.     model.add(Convolution2D(32, 3, 3, border_mode='same',
26.                             input_shape=X_train.shape[1:]))
27.     model.add(Activation('relu'))
28.     model.add(Convolution2D(32, 3, 3))
29.     model.add(Activation('relu'))
30.     model.add(MaxPooling2D(pool_size=(2, 2)))
31.     model.add(Dropout(0.5))
32.
33.
34.     if conditional({{choice(['two', 'four'])}}) == 'four':
35.         model.add(Convolution2D(64, 3, 3, border_mode='same'))
36.         model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
37.         model.add(Convolution2D(64, 3, 3))
38.         model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
39.         model.add(MaxPooling2D(pool_size=(2, 2)))
40.         model.add(Dropout({{uniform(0, 1)}}))
41.
42.     model.add(Flatten())
43.     model.add(Dense(512))
44.     model.add(Activation('relu'))
45.     model.add(Dropout({{uniform(0, 1)}}))
46.     model.add(Dense(10))
47.     model.add(Activation('softmax'))
48.
49.     # train the model
50.     model.compile(loss='categorical_crossentropy',
51.                   optimizer='rmsprop',
52.                   metrics=['accuracy'])
53.
54.     # fit the model
55.     model.fit_generator(datagen.flow(X_train, Y_train,
56.                         batch_size=batch_size),
57.                         samples_per_epoch=X_train.shape[0],
58.                         nb_epoch=nb_epoch,
59.                         validation_data=(X_test, Y_test))
60.
61.     return {'acc': -acc, 'status': STATUS_OK, 'model': model}
```

A.2 Optimal Batch Size

```
1.  def model(datagen, X_train, Y_train, X_test, Y_test):
2.  
3.      # number of epochs
4.      nb_epoch = 30
5.  
6.      # batch size
7.      batch_size = {{choice([16, 32, 64, 128, 256, 512])}}
8.  
9.      # input image dimensions
10.     img_rows, img_cols = 32, 32
11. 
12.     # image channels
13.     img_channels = 3
14. 
15.     model = Sequential()
16. 
17.     # Convolution2D: Convolutional Layer
18.     # Activation: Activation Layer
19.     # MaxPooling2D: MaxPooling Layer
20.     # Dropout: Dropout Layer
21.     # optimizer: Optimizer Method
22. 
23.     model = Sequential()
24. 
25.     model.add(Convolution2D(32, 3, 3, border_mode='same',
26.                             input_shape=X_train.shape[1:]))
27.     model.add(Activation('relu'))
28.     model.add(Convolution2D(32, 3, 3))
29.     model.add(Activation('relu'))
30.     model.add(MaxPooling2D(pool_size=(2, 2)))
31.     model.add(Dropout(0.5))
32. 
33.     model.add(Convolution2D(64, 3, 3, border_mode='same'))
34.     model.add(Activation('relu'))
35.     model.add(Convolution2D(64, 3, 3))
36.     model.add(Activation('relu'))
37.     model.add(MaxPooling2D(pool_size=(2, 2)))
38.     model.add(Dropout(0.5))
39. 
40.     model.add(Flatten())
41.     model.add(Dense(512))
42.     model.add(Activation('relu'))
43.     model.add(Dropout({{uniform(0, 1)}}))
44.     model.add(Dense(10))
45.     model.add(Activation('softmax'))
46. 
47.     # train the model
48.     model.compile(loss='categorical_crossentropy',
49.                   optimizer='rmsprop',
50.                   metrics=['accuracy'])
51. 
52.     # fit the model
53.     model.fit_generator(datagen.flow(X_train, Y_train,
54.                         batch_size=batch_size),
55.                         samples_per_epoch=X_train.shape[0],
56.                         nb_epoch=nb_epoch,
57.                         validation_data=(X_test, Y_test))
58. 
59.     return {'acc': -acc, 'status': STATUS_OK, 'model': model}
```

A.3 Optimal Activation Functions

```
1.  def model(datagen, X_train, Y_train, X_test, Y_test):
2.
3.      # number of epochs
4.      nb_epoch = 30
5.
6.      # batch size
7.      batch_size = 64
8.
9.      # input image dimensions
10.     img_rows, img_cols = 32, 32
11.
12.     # image channels
13.     img_channels = 3
14.
15.     model = Sequential()
16.
17.     # Convolution2D: Convolutional Layer
18.     # Activation: Activation Layer
19.     # MaxPooling2D: MaxPooling Layer
20.     # Dropout: Dropout Layer
21.     # optimizer: Optimizer Method
22.
23.     model.add(Convolution2D(32, 3, 3, border_mode='same',
24.                             input_shape=X_train.shape[1:]))
25.     model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
26.     model.add(Convolution2D(32, 3, 3))
27.     model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
28.     model.add(MaxPooling2D(pool_size=(2, 2)))
29.     model.add(Dropout(0.5))
30.
31.     model.add(Convolution2D(64, 3, 3, border_mode='same'))
32.     model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
33.     model.add(Convolution2D(64, 3, 3))
34.     model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
35.     model.add(MaxPooling2D(pool_size=(2, 2)))
36.     model.add(Dropout(0.5))
37.
38.     model.add(Flatten())
39.     model.add(Dense(512))
40.     model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
41.     model.add(Dropout(0.5))
42.     model.add(Dense(10))
43.     model.add(Activation('softmaxs'))
44.
45.     # train the model
46.     model.compile(loss='categorical_crossentropy',
47.                   optimizer={{choice(['rmsprop', 'adam', 'sgd'])}},
48.                   metrics=['accuracy'])
49.
50.     # fit the model
51.     model.fit_generator(datagen.flow(X_train, Y_train,
52.                         batch_size=batch_size),
53.                         samples_per_epoch=X_train.shape[0],
54.                         nb_epoch=nb_epoch,
55.                         validation_data=(X_test, Y_test))
56.
57.     return {'acc': -acc, 'status': STATUS_OK, 'model': model}
```

A.4 Optimal Dropout Values

```
1.  def model(datagen, X_train, Y_train, X_test, Y_test):
2.  
3.      # number of epochs
4.      nb_epoch = 30
5.  
6.      # batch size
7.      batch_size = 64
8.  
9.      # input image dimensions
10.     img_rows, img_cols = 32, 32
11. 
12.     # image channels
13.     img_channels = 3
14. 
15.     model = Sequential()
16. 
17.     # Convolution2D: Convolutional Layer
18.     # Activation: Activation Layer
19.     # MaxPooling2D: MaxPooling Layer
20.     # Dropout: Dropout Layer
21.     # optimizer: Optimizer Method
22. 
23.     model = Sequential()
24. 
25.     model.add(Convolution2D(32, 3, 3, border_mode='same',
26.                             input_shape=X_train.shape[1:]))
27.     model.add(Activation('relu'))
28.     model.add(Convolution2D(32, 3, 3))
29.     model.add(Activation('relu'))
30.     model.add(MaxPooling2D(pool_size=(2, 2)))
31.     model.add(Dropout({{uniform(0, 1)}}))
32. 
33.     model.add(Convolution2D(64, 3, 3, border_mode='same'))
34.     model.add(Activation('relu'))
35.     model.add(Convolution2D(64, 3, 3))
36.     model.add(Activation('relu'))
37.     model.add(MaxPooling2D(pool_size=(2, 2)))
38.     model.add(Dropout({{uniform(0, 1)}}))
39. 
40.     model.add(Flatten())
41.     model.add(Dense(512))
42.     model.add(Activation('relu'))
43.     model.add(Dropout({{uniform(0, 1)}}))
44.     model.add(Dense(10))
45.     model.add(Activation('softmax'))
46. 
47.     # train the model
48.     model.compile(loss='categorical_crossentropy',
49.                   optimizer='rmsprop',
50.                   metrics=['accuracy'])
51. 
52.     # fit the model
53.     model.fit_generator(datagen.flow(X_train, Y_train,
54.                         batch_size=batch_size),
55.                         samples_per_epoch=X_train.shape[0],
56.                         nb_epoch=nb_epoch,
57.                         validation_data=(X_test, Y_test))
58. 
59.     return {'acc': -acc, 'status': STATUS_OK, 'model': model}
```

A.5. Optimal Optimizer Method

```
1.  def model(datagen, X_train, Y_train, X_test, Y_test):
2.  
3.      # number of epochs
4.      nb_epoch = 30
5.  
6.      # batch size
7.      batch_size = 64
8.  
9.      # input image dimensions
10.     img_rows, img_cols = 32, 32
11. 
12.     # image channels
13.     img_channels = 3
14. 
15.     model = Sequential()
16. 
17.     # Convolution2D: Convolutional Layer
18.     # Activation: Activation Layer
19.     # MaxPooling2D: MaxPooling Layer
20.     # Dropout: Dropout Layer
21.     # optimizer: Optimizer Method
22. 
23.     model = Sequential()
24. 
25.     model.add(Convolution2D(32, 3, 3, border_mode='same',
26.                             input_shape=X_train.shape[1:]))
27.     model.add(Activation('relu'))
28.     model.add(Convolution2D(32, 3, 3))
29.     model.add(Activation('relu'))
30.     model.add(MaxPooling2D(pool_size=(2, 2)))
31.     model.add(Dropout(0.5))
32. 
33.     model.add(Convolution2D(64, 3, 3, border_mode='same'))
34.     model.add(Activation('relu'))
35.     model.add(Convolution2D(64, 3, 3))
36.     model.add(Activation('relu'))
37.     model.add(MaxPooling2D(pool_size=(2, 2)))
38.     model.add(Dropout(0.5))
39. 
40.     model.add(Flatten())
41.     model.add(Dense(512))
42.     model.add(Activation('relu'))
43.     model.add(Dropout({{uniform(0, 1)}}))
44.     model.add(Dense(10))
45.     model.add(Activation('softmax'))
46. 
47.     # train the model
48.     model.compile(loss='categorical_crossentropy',
49.                   optimizer={{choice(['rmsprop', 'adam', 'sgd','adagrad','adadel
    ta','adamax','nadam'])}},
50.                   metrics=['accuracy'])
51. 
52.     # fit the model
53.     model.fit_generator(datagen.flow(X_train, Y_train,
54.                         batch_size=batch_size),
55.                         samples_per_epoch=X_train.shape[0],
56.                         nb_epoch=nb_epoch,
57.                         validation_data=(X_test, Y_test))
58. 
59.     return {'acc': -acc, 'status': STATUS_OK, 'model': model}
```

A.6 Optimal Values for All Chosen Hyperparameters

```
1.  def model(datagen, X_train, Y_train, X_test, Y_test):
2.
3.      # number of epochs
4.      nb_epoch = 30
5.
6.      # batch size
7.      batch_size = {{choice([16, 32, 64, 128, 256, 512])}}
8.
9.      # input image dimensions
10.     img_rows, img_cols = 32, 32
11.
12.     # image channels
13.     img_channels = 3
14.
15.     model = Sequential()
16.
17.     # Convolution2D: Convolutional Layer
18.     # Activation: Activation Layer
19.     # MaxPooling2D: MaxPooling Layer
20.     # Dropout: Dropout Layer
21.     # optimizer: Optimizer Method
22.
23.     model = Sequential()
24.
25.     model.add(Convolution2D(32, 3, 3, border_mode='same',
26.                             input_shape=X_train.shape[1:]))
27.     model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
28.     model.add(Convolution2D(32, 3, 3))
29.     model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
30.     model.add(MaxPooling2D(pool_size=(2, 2)))
31.     model.add(Dropout({{uniform(0, 1)}}))
32.
33.     if conditional({{choice(['two', 'four'])}}) == 'four':
34.         model.add(Convolution2D(64, 3, 3, border_mode='same'))
35.         model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
36.         model.add(Convolution2D(64, 3, 3))
37.         model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
38.         model.add(MaxPooling2D(pool_size=(2, 2)))
39.         model.add(Dropout({{uniform(0, 1)}}))
40.
41.     model.add(Flatten())
42.     model.add(Dense(512))
43.     model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
44.     model.add(Dropout({{uniform(0, 1)}}))
45.     model.add(Dense(10))
46.     model.add(Activation('softmax'))
47.
48.     # train the model
49.     model.compile(loss='categorical_crossentropy',
50.                   optimizer={{choice(['rmsprop', 'adam', 'sgd'])}},
51.                   metrics=['accuracy'])
52.
53.     # fit the model
54.     model.fit_generator(datagen.flow(X_train, Y_train,
55.                         batch_size=batch_size),
56.                         samples_per_epoch=X_train.shape[0],
57.                         nb_epoch=nb_epoch,
58.                         validation_data=(X_test, Y_test))
59.
60.     return {'acc': -acc, 'status': STATUS_OK, 'model': model}
```

B. Source Code of Hyperparameter Optimization for Complex CNN

B.1 Optimal Number of Convolutional Layers

```
1.  def model(datagen, X_train, Y_train, X_test, Y_test):
2.
3.      # number of epochs
4.      nb_epoch = 30
5.
6.      # batch size
7.      batch_size = 64
8.
9.      # input image dimensions
10.     img_rows, img_cols = 32, 32
11.
12.     # image channels
13.     img_channels = 3
14.
15.     model = Sequential()
16.
17.     # Convolution2D: Convolutional Layer
18.     # Activation: Activation Layer
19.     # MaxPooling2D: MaxPooling Layer
20.     # Dropout: Dropout Layer
21.     # optimizer: Optimizer Method
22.
23.     model = Sequential()
24.
25.     model.add(Convolution2D(32, 3, 3, border_mode='same',
26.                             input_shape=X_train.shape[1:]))
27.     model.add(Activation('relu'))
28.     model.add(Convolution2D(32, 3, 3))
29.     model.add(Activation('relu'))
30.     model.add(MaxPooling2D(pool_size=(2, 2)))
31.     model.add(Dropout(0.5))
32.
33.     model.add(Convolution2D(64, 3, 3, border_mode='same'))
34.     model.add(Activation('relu'))
35.     model.add(Convolution2D(64, 3, 3))
36.     model.add(Activation('relu'))
37.     model.add(Dropout(0.5))
38.
39.     if conditional({{choice(['two', 'three'])}}) == 'three':
40.         model.add(Convolution2D(128, 3, 3, border_mode='same'))
41.         model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
42.         model.add(Convolution2D(128, 3, 3))
43.         model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
44.         model.add(MaxPooling2D(pool_size=(2, 2)))
45.         model.add(Dropout({{uniform(0, 1)}}))
46.
47.     model.add(Convolution2D(256, 3, 3, border_mode='same'))
48.     model.add(Activation('relu'))
49.     model.add(Convolution2D(256, 3, 3))
50.     model.add(Activation('relu'))
51.     model.add(Dropout(0.5))
52.
53.     model.add(Activation('relu'))
54.     model.add(Activation('relu'))
55.     model.add(Dropout(0.5))
56.
57.     model.add(Flatten())
58.     model.add(Dense(512))
59.     model.add(Activation('relu'))
60.     model.add(Dropout({{uniform(0, 1)}}))
61.     model.add(Dense(10))
62.     model.add(Activation('softmax'))
63.
64.     # train the model
65.     model.compile(loss='categorical_crossentropy',
66.                   optimizer='rmsprop',
67.                   metrics=['accuracy'])
68.
69.     # fit the model
70.     model.fit_generator(datagen.flow(X_train, Y_train,
71.                         batch_size=batch_size),
72.                         samples_per_epoch=X_train.shape[0],
73.                         nb_epoch=nb_epoch,
74.                         validation_data=(X_test, Y_test))
75.
76.     return {'acc': -acc, 'status': STATUS_OK, 'model': model}
```

B.2 Optimal Batch Size

```
1.  def model(datagen, X_train, Y_train, X_test, Y_test):
2.  
3.      # number of epochs
4.      nb_epoch = 30
5.  
6.      # batch size
7.      batch_size = {{choice([16, 32, 64, 128, 256, 512])}}
8.  
9.      # input image dimensions
10.     img_rows, img_cols = 32, 32
11. 
12.     # image channels
13.     img_channels = 3
14. 
15.     model = Sequential()
16. 
17.     # Convolution2D: Convolutional Layer
18.     # Activation: Activation Layer
19.     # MaxPooling2D: MaxPooling Layer
20.     # Dropout: Dropout Layer
21.     # optimizer: Optimizer Method
22. 
23.     model = Sequential()
24. 
25.     model.add(Convolution2D(32, 3, 3, border_mode='same',
26.                             input_shape=X_train.shape[1:]))
27.     model.add(Activation('relu'))
28.     model.add(Convolution2D(32, 3, 3))
29.     model.add(Activation('relu'))
30.     model.add(MaxPooling2D(pool_size=(2, 2)))
31.     model.add(Dropout(0.5))
32. 
33.     model.add(Convolution2D(64, 3, 3, border_mode='same'))
34.     model.add(Activation('relu'))
35.     model.add(Convolution2D(64, 3, 3))
36.     model.add(Activation('relu'))
37.     model.add(Dropout(0.5))
38. 
39.     model.add(Convolution2D(128, 3, 3, border_mode='same'))
40.     model.add(Activation('relu'))
41.     model.add(Convolution2D(128, 3, 3))
42.     model.add(Activation('relu'))
43.     model.add(MaxPooling2D(pool_size=(2, 2)))
44.     model.add(Dropout(0.5))
45. 
46.     model.add(Convolution2D(256, 3, 3, border_mode='same'))
47.     model.add(Activation('relu'))
48.     model.add(Convolution2D(256, 3, 3))
49.     model.add(Activation('relu'))
50.     model.add(Dropout(0.5))
51. 
52.     model.add(Activation('relu'))
53.     model.add(Activation('relu'))
54.     model.add(Dropout(0.5))
55. 
56.     model.add(Flatten())
57.     model.add(Dense(512))
58.     model.add(Activation('relu'))
59.     model.add(Dropout({{uniform(0, 1)}}))
60.     model.add(Dense(10))
61.     model.add(Activation('softmax'))
62. 
63.     # train the model
64.     model.compile(loss='categorical_crossentropy',
65.                   optimizer='rmsprop',
66.                   metrics=['accuracy'])
67. 
68.     # fit the model
69.     model.fit_generator(datagen.flow(X_train, Y_train,
70.                         batch_size=batch_size),
71.                         samples_per_epoch=X_train.shape[0],
72.                         nb_epoch=nb_epoch,
73.                         validation_data=(X_test, Y_test))
74. 
75.     return {'acc': -acc, 'status': STATUS_OK, 'model': model}
```

B.3 Optimal Activation Functions

```
1.  def model(datagen, X_train, Y_train, X_test, Y_test):
2.
3.      # number of epochs
4.      nb_epoch = 30
5.
6.      # batch size
7.      batch_size = 64
8.
9.      # input image dimensions
10.     img_rows, img_cols = 32, 32
11.
12.     # image channels
13.     img_channels = 3
14.
15.     model = Sequential()
16.
17.     # Convolution2D: Convolutional Layer
18.     # Activation: Activation Layer
19.     # MaxPooling2D: MaxPooling Layer
20.     # Dropout: Dropout Layer
21.     # optimizer: Optimizer Method
22.
23.     model.add(Convolution2D(32, 3, 3, border_mode='same',
24.                             input_shape=X_train.shape[1:]))
25.     model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
26.     model.add(Convolution2D(32, 3, 3))
27.     model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
28.     model.add(MaxPooling2D(pool_size=(2, 2)))
29.     model.add(Dropout(0.5))
30.
31.     model.add(Convolution2D(64, 3, 3, border_mode='same'))
32.     model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
33.     model.add(Convolution2D(64, 3, 3))
34.     model.add(Dropout(0.5))
35.
36.     model.add(Convolution2D(128, 3, 3, border_mode='same'))
37.     model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
38.     model.add(Convolution2D(128, 3, 3))
39.     model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
40.     model.add(MaxPooling2D(pool_size=(2, 2)))
41.     model.add(Dropout(0.5))
42.
43.     model.add(Convolution2D(256, 3, 3, border_mode='same'))
44.     model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
45.     model.add(Convolution2D(256, 3, 3))
46.     model.add(Dropout(0.5))
47.
48.     model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
49.     model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
50.     model.add(Dropout(0.5))
51.
52.     model.add(Flatten())
53.     model.add(Dense(512))
54.     model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
55.     model.add(Dropout(0.5))
56.     model.add(Dense(10))
57.     model.add(Activation('softmax'))
58.
59.     # train the model
60.     model.compile(loss='categorical_crossentropy',
61.                   optimizer={{choice(['rmsprop', 'adam', 'sgd'])}},
62.                   metrics=['accuracy'])
63.
64.     # fit the model
65.     model.fit_generator(datagen.flow(X_train, Y_train,
66.                         batch_size=batch_size),
67.                         samples_per_epoch=X_train.shape[0],
68.                         nb_epoch=nb_epoch,
69.                         validation_data=(X_test, Y_test))
70.
71.     return {'acc': -acc, 'status': STATUS_OK, 'model': model}
```

B.4 Optimal Dropout Values

```
1.  def model(datagen, X_train, Y_train, X_test, Y_test):
2.
3.      # number of epochs
4.      nb_epoch = 30
5.
6.      # batch size
7.      batch_size = 64
8.
9.      # input image dimensions
10.     img_rows, img_cols = 32, 32
11.
12.     # image channels
13.     img_channels = 3
14.
15.     model = Sequential()
16.
17.     # Convolution2D: Convolutional Layer
18.     # Activation: Activation Layer
19.     # MaxPooling2D: MaxPooling Layer
20.     # Dropout: Dropout Layer
21.     # optimizer: Optimizer Method
22.
23.     model = Sequential()
24.
25.     model.add(Convolution2D(32, 3, 3, border_mode='same',
26.                             input_shape=X_train.shape[1:]))
27.     model.add(Activation('relu'))
28.     model.add(Convolution2D(32, 3, 3))
29.     model.add(Activation('relu'))
30.     model.add(MaxPooling2D(pool_size=(2, 2)))
31.     model.add(Dropout({{uniform(0, 1)}}))
32.
33.     model.add(Convolution2D(64, 3, 3, border_mode='same'))
34.     model.add(Activation('relu'))
35.     model.add(Convolution2D(64, 3, 3))
36.     model.add(Activation('relu'))
37.     model.add(Dropout({{uniform(0, 1)}}))
38.
39.     model.add(Convolution2D(128, 3, 3, border_mode='same'))
40.     model.add(Activation('relu'))
41.     model.add(Convolution2D(128, 3, 3))
42.     model.add(Activation('relu'))
43.     model.add(MaxPooling2D(pool_size=(2, 2)))
44.     model.add(Dropout({{uniform(0, 1)}}))
45.
46.     model.add(Convolution2D(256, 3, 3, border_mode='same'))
47.     model.add(Activation('relu'))
48.     model.add(Convolution2D(256, 3, 3))
49.     model.add(Activation('relu'))
50.     model.add(Dropout({{uniform(0, 1)}}))
51.
52.     model.add(Activation('relu'))
53.     model.add(Activation('relu'))
54.     model.add(Dropout({{uniform(0, 1)}}))
55.
56.     model.add(Flatten())
57.     model.add(Dense(512))
58.     model.add(Activation('relu'))
59.     model.add(Dropout({{uniform(0, 1)}}))
60.     model.add(Dense(10))
61.     model.add(Activation('softmax'))
62.
63.     # train the model
64.     model.compile(loss='categorical_crossentropy',
65.                   optimizer='rmsprop',
66.                   metrics=['accuracy'])
67.
68.     # fit the model
69.     model.fit_generator(datagen.flow(X_train, Y_train,
70.                         batch_size=batch_size),
71.                         samples_per_epoch=X_train.shape[0],
72.                         nb_epoch=nb_epoch,
73.                         validation_data=(X_test, Y_test))
74.
75.     return {'acc': -acc, 'status': STATUS_OK, 'model': model}
```

B.5 Optimal Optimizer Method

```
1.  def model(datagen, X_train, Y_train, X_test, Y_test):
2.
3.      # number of epochs
4.      nb_epoch = 30
5.
6.      # batch size
7.      batch_size = 64
8.
9.      # input image dimensions
10.     img_rows, img_cols = 32, 32
11.
12.     # image channels
13.     img_channels = 3
14.
15.     model = Sequential()
16.
17.     # Convolution2D: Convolutional Layer
18.     # Activation: Activation Layer
19.     # MaxPooling2D: MaxPooling Layer
20.     # Dropout: Dropout Layer
21.     # optimizer: Optimizer Method
22.
23.     model = Sequential()
24.
25.     model.add(Convolution2D(32, 3, 3, border_mode='same',
26.                             input_shape=X_train.shape[1:]))
27.     model.add(Activation('relu'))
28.     model.add(Convolution2D(32, 3, 3))
29.     model.add(Activation('relu'))
30.     model.add(MaxPooling2D(pool_size=(2, 2)))
31.     model.add(Dropout(0.5))
32.
33.     model.add(Convolution2D(64, 3, 3, border_mode='same'))
34.     model.add(Activation('relu'))
35.     model.add(Convolution2D(64, 3, 3))
36.     model.add(Activation('relu'))
37.     model.add(Dropout(0.5))
38.
39.     model.add(Convolution2D(128, 3, 3, border_mode='same'))
40.     model.add(Activation('relu'))
41.     model.add(Convolution2D(128, 3, 3))
42.     model.add(Activation('relu'))
43.     model.add(MaxPooling2D(pool_size=(2, 2)))
44.     model.add(Dropout(0.5))
45.
46.     model.add(Convolution2D(256, 3, 3, border_mode='same'))
47.     model.add(Activation('relu'))
48.     model.add(Convolution2D(256, 3, 3))
49.     model.add(Activation('relu'))
50.     model.add(Dropout(0.5))
51.
52.     model.add(Activation('relu'))
53.     model.add(Activation('relu'))
54.     model.add(Dropout(0.5))
55.
56.     model.add(Flatten())
57.     model.add(Dense(512))
58.     model.add(Activation('relu'))
59.     model.add(Dropout({{uniform(0, 1)}}))
60.     model.add(Dense(10))
61.     model.add(Activation('softmax'))
62.
63.     # train the model
64.     model.compile(loss='categorical_crossentropy',
65.                   optimizer={{choice(['rmsprop', 'adam', 'sgd','adagrad','adadel
    ta','adamax','nadam'])}},
66.                   metrics=['accuracy'])
67.
68.     # fit the model
69.     model.fit_generator(datagen.flow(X_train, Y_train,
70.                         batch_size=batch_size),
71.                         samples_per_epoch=X_train.shape[0],
72.                         nb_epoch=nb_epoch,
73.                         validation_data=(X_test, Y_test))
74.
75.     return {'acc': -acc, 'status': STATUS_OK, 'model': model}
```

B.6 Optimal Values for All Chosen Hyperparameters

```
1.  def model(datagen, X_train, Y_train, X_test, Y_test):
2.
3.      # number of epochs
4.      nb_epoch = 30
5.
6.      # batch size
7.      batch_size = {{choice([16, 32, 64, 128, 256, 512])}}
8.
9.      # input image dimensions
10.     img_rows, img_cols = 32, 32
11.
12.     # image channels
13.     img_channels = 3
14.
15.     model = Sequential()
16.
17.     # Convolution2D: Convolutional Layer
18.     # Activation: Activation Layer
19.     # MaxPooling2D: MaxPooling Layer
20.     # Dropout: Dropout Layer
21.     # optimizer: Optimizer Method
22.
23.     model = Sequential()
24.
25.     model.add(Convolution2D(32, 3, 3, border_mode='same',
26.                             input_shape=X_train.shape[1:]))
27.     model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
28.     model.add(Convolution2D(32, 3, 3))
29.     model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
30.     model.add(MaxPooling2D(pool_size=(2, 2)))
31.     model.add(Dropout({{uniform(0, 1)}}))
32.
33.     model.add(Convolution2D(64, 3, 3, border_mode='same'))
34.     model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
35.     model.add(Convolution2D(64, 3, 3))
36.     model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
37.     model.add(Dropout({{uniform(0, 1)}}))
38.
39.     if conditional({{choice(['two', 'three'])}}) == 'three':
40.         model.add(Convolution2D(128, 3, 3, border_mode='same'))
41.         model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
42.         model.add(Convolution2D(128, 3, 3))
43.         model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
44.         model.add(MaxPooling2D(pool_size=(2, 2)))
45.         model.add(Dropout({{uniform(0, 1)}}))
46.
47.     model.add(Convolution2D(256, 3, 3, border_mode='same'))
48.     model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
49.     model.add(Convolution2D(256, 3, 3))
50.     model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
51.     model.add(Dropout({{uniform(0, 1)}}))
52.
53.     model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
54.     model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
55.     model.add(Dropout({{uniform(0, 1)}}))
56.
57.     model.add(Flatten())
58.     model.add(Dense(512))
59.     model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
60.     model.add(Dropout({{uniform(0, 1)}}))
61.     model.add(Dense(10))
62.     model.add(Activation('softmax'))
63.
64.     # train the model
65.     model.compile(loss='categorical_crossentropy',
66.                   optimizer={{choice(['rmsprop', 'adam', 'sgd'])}},
67.                   metrics=['accuracy'])
68.
69.     # fit the model
70.     model.fit_generator(datagen.flow(X_train, Y_train,
71.                         batch_size=batch_size),
72.                         samples_per_epoch=X_train.shape[0],
73.                         nb_epoch=nb_epoch,
74.                         validation_data=(X_test, Y_test))
75.
76.     return {'acc': -acc, 'status': STATUS_OK, 'model': model}
```

C. Accuracy and Loss of All Chosen Hyperparameters for Simple CNN

C.1 Random Search

The highlighted accuracy and loss represent those of the optimized CNN.

Epoch	Accuracy	Loss	Epoch	Accuracy	Loss	Epoch	Accuracy	Loss
1	0.09891	2.31080	31	0.21733	2.29417	61	0.09915	2.41330
2	0.10033	2.30320	32	0.25780	2.29084	62	0.10026	2.30430
3	0.09937	2.30280	33	0.13542	2.29010	63	0.09872	2.30437
4	0.09933	2.30280	34	0.10011	2.28973	64	0.10164	2.30378
5	0.09967	2.30280	35	0.10000	2.27842	65	0.10991	2.29863
6	0.09812	2.30260	36	0.09981	2.26424	66	0.18216	2.16874
7	0.09900	2.30280	37	0.10000	2.25011	67	0.21626	2.01538
8	0.10198	2.30260	38	0.10025	2.22462	68	0.23192	1.95144
9	0.09770	2.30270	39	0.09991	2.23635	69	0.24180	1.90116
10	0.10040	2.30280	40	0.10037	2.22410	70	0.25516	1.87065
11	0.10023	2.30260	41	0.09951	2.20680	71	0.26710	1.83641
12	0.09865	2.30260	42	0.10025	2.20157	72	0.27383	1.80283
13	0.10040	2.30260	43	0.10011	2.19560	73	0.29790	1.74286
14	0.09626	2.30270	44	0.09990	2.18364	74	0.30597	1.72828
15	0.09813	2.30260	45	0.10017	2.16379	75	0.32636	1.68752
16	0.10068	2.30260	46	0.10000	2.14468	76	0.34399	1.64995
17	0.10062	2.30260	47	0.10031	2.13250	77	0.35650	1.62940
18	0.10159	2.30270	48	0.09986	2.11923	78	0.36093	1.58539
19	0.09962	2.30260	49	0.10000	2.10571	79	0.38736	1.56155
20	0.09934	2.30260	50	0.10000	2.09434	80	0.40355	1.53579
21	0.09960	2.30270	51	0.09997	2.08857	81	0.41974	1.50967
22	0.09853	2.30260	52	0.10028	2.06330	82	0.43740	1.47070
23	0.10067	2.30260	53	0.09980	2.05235	83	0.45286	1.44210
24	0.10112	2.30260	54	0.10022	2.04317	84	0.48681	1.41835
25	0.09928	2.30260	55	0.09987	2.04000	85	0.49950	1.39453
26	0.09874	2.30260	56	0.10010	2.03635	86	0.48014	1.37152
27	0.10216	2.30260	57	0.10000	2.02520	87	0.47372	1.36059
28	0.09976	2.30260	58	0.09986	2.01943	88	0.46920	1.35642
29	0.09823	2.30260	59	0.10026	2.00156	89	0.43730	1.34531
30	0.09960	2.30260	60	0.10031	1.99837	90	0.43470	1.33190

C.2 Tree-Structured Parzen Estimator

The highlighted accuracy and loss represent those of the optimized CNN.

Epoch	Accuracy	Loss	Epoch	Accuracy	Loss	Epoch	Accuracy	Loss
1	0.09931	2.31082	31	0.18100	2.35055	61	0.10010	2.42070
2	0.10240	2.30312	32	0.22180	2.34521	62	0.09954	2.30400
3	0.10115	2.30370	33	0.11734	2.32725	63	0.09993	2.30400
4	0.09990	2.30286	34	0.10000	2.30865	64	0.09914	2.30390
5	0.10000	2.30274	35	0.09990	2.28042	65	0.14324	2.25770
6	0.10000	2.30287	36	0.10000	2.27725	66	0.19236	2.09740
7	0.09955	2.30273	37	0.09990	2.26840	67	0.20417	1.99670
8	0.10026	2.30270	38	0.10000	2.25270	68	0.20845	1.96550
9	0.10027	2.30264	39	0.10000	2.24948	69	0.21134	1.94620
10	0.09946	2.30267	40	0.10010	2.23729	70	0.21362	1.92920
11	0.09945	2.30276	41	0.10010	2.22660	71	0.21470	1.92160
12	0.10000	2.30267	42	0.09991	2.21965	72	0.21580	1.91080
13	0.10087	2.30266	43	0.09986	2.20586	73	0.22717	1.90360
14	0.09995	2.30278	44	0.10044	2.19687	74	0.23959	1.89390
15	0.10317	2.30269	45	0.09960	2.19650	75	0.24381	1.87940
16	0.09985	2.30272	46	0.10035	2.18466	76	0.24624	1.85170
17	0.09684	2.30265	47	0.10010	2.17295	77	0.25367	1.83140
18	0.10163	2.30264	48	0.10000	2.16277	78	0.27250	1.80360
19	0.10055	2.30264	49	0.09990	2.15014	79	0.28783	1.78780
20	0.10074	2.30260	50	0.10000	2.14676	80	0.30141	1.75200
21	0.10319	2.30260	51	0.10020	2.13982	81	0.31038	1.72170
22	0.09948	2.30260	52	0.09973	2.12667	82	0.32543	1.69650
23	0.10150	2.30260	53	0.10010	2.11053	83	0.33940	1.65560
24	0.09965	2.30260	54	0.09991	2.10362	84	0.33070	1.67120
25	0.10020	2.30260	55	0.10030	2.09547	85	0.32061	1.68240
26	0.09944	2.30260	56	0.09981	2.08465	86	0.31157	1.68230
27	0.09824	2.30260	57	0.10035	2.07395	87	0.31117	1.69020
28	0.10186	2.30260	58	0.10000	2.06236	88	0.30853	1.69280
29	0.10010	2.30260	59	0.09972	2.05560	89	0.30562	1.69010
30	0.09957	2.30270	60	0.10000	2.05000	90	0.29493	1.69540

D. Accuracy and Loss of All Chosen Hyperparameters for Complex CNN

D.1 Random Search

The highlighted accuracy and loss represent those of the optimized CNN.

Epoch	Accuracy	Loss	Epoch	Accuracy	Loss	Epoch	Accuracy	Loss
1	0.09938	2.42658	31	0.19104	2.41047	61	0.10010	2.44561
2	0.10011	2.43650	32	0.21495	2.36046	62	0.09802	2.31954
3	0.10011	2.45695	33	0.12856	2.34761	63	0.11734	2.23460
4	0.09949	2.42951	34	0.10000	2.32657	64	0.11825	2.17453
5	0.10027	2.42538	35	0.09990	2.29532	65	0.14334	2.12025
6	0.10009	2.41466	36	0.10000	2.27340	66	0.19256	2.08750
7	0.10009	2.42569	37	0.09990	2.26831	67	0.21546	2.04941
8	0.09955	2.41270	38	0.11251	2.25380	68	0.22821	2.03094
9	0.10007	2.40467	39	0.10000	2.24568	69	0.23100	2.03338
10	0.10011	2.40691	40	0.10010	2.23920	70	0.23592	2.02560
11	0.10041	2.40481	41	0.10010	2.22015	71	0.24438	2.01129
12	0.09931	2.40548	42	0.09980	2.21172	72	0.24970	2.00325
13	0.09987	2.40815	43	0.09185	2.20435	73	0.24794	1.99818
14	0.10055	2.40548	44	0.10196	2.19546	74	0.25011	1.98547
15	0.09977	2.40464	45	0.09943	2.19340	75	0.25318	1.98547
16	0.09985	2.40465	46	0.11541	2.18620	76	0.25156	1.97825
17	0.10039	2.40454	47	0.10324	2.17496	77	0.26431	1.96360
18	0.09955	2.40451	48	0.10000	2.15915	78	0.27251	1.95671
19	0.10035	2.40453	49	0.09351	2.15001	79	0.28783	1.95824
20	0.09953	2.40453	50	0.10000	2.14954	80	0.29141	1.94604
21	0.1012	2.40453	51	0.10342	2.13731	81	0.31031	1.93100
22	0.09929	2.40453	52	0.09831	2.12356	82	0.32815	1.92436
23	0.09937	2.40453	53	0.10013	2.11140	83	0.32916	1.92120
24	0.10021	2.40453	54	0.09835	2.10012	84	0.33541	1.91578
25	0.10074	2.40453	55	0.10541	2.09543	85	0.33573	1.90258
26	0.10031	2.40453	56	0.09832	2.08843	86	0.33157	1.91541
27	0.10017	2.40453	57	0.11454	2.07031	87	0.32168	1.91682
28	0.09865	2.40453	58	0.10000	2.06980	88	0.31676	1.92046
29	0.101	2.40453	59	0.09972	2.05539	89	0.30165	1.93695
30	0.09927	2.40453	60	0.10000	2.05593	90	0.30770	1.93043

D.2 Tree-Structured Parzen Estimator

The highlighted accuracy and loss represent those of the optimized CNN.

Epoch	Accuracy	Loss	Epoch	Accuracy	Loss	Epoch	Accuracy	Loss
1	0.10008	2.42560	31	0.19359	2.29417	61	0.09781	2.44300
2	0.09997	2.42570	32	0.17019	2.29084	62	0.10438	2.42542
3	0.10015	2.42560	33	0.14958	2.29010	63	0.09917	2.40318
4	0.09995	2.42580	34	0.12951	2.28973	64	0.10031	2.38605
5	0.10011	2.42590	35	0.10169	2.27842	65	0.10991	2.36710
6	0.09989	2.42570	36	0.09982	2.26424	66	0.18491	2.33432
7	0.10011	2.42580	37	0.11940	2.25011	67	0.20718	2.30367
8	0.09975	2.42570	38	0.10025	2.22462	68	0.22954	2.26597
9	0.09981	2.42570	39	0.08291	2.23635	69	0.24195	2.23601
10	0.10043	2.42580	40	0.07041	2.22410	70	0.25015	2.20682
11	0.09967	2.42570	41	0.07041	2.20680	71	0.26549	2.17938
12	0.10029	2.42560	42	0.07924	2.20157	72	0.27941	2.13828
13	0.09961	2.42570	43	0.08043	2.19560	73	0.29491	2.09520
14	0.10078	2.42580	44	0.08710	2.18364	74	0.30963	2.06864
15	0.09923	2.42580	45	0.08495	2.16379	75	0.32094	2.04682
16	0.09971	2.42560	46	0.08596	2.14468	76	0.34429	2.00577
17	0.10096	2.42560	47	0.07437	2.13250	77	0.35058	1.96591
18	0.09867	2.42568	48	0.08002	2.11923	78	0.36283	1.94638
19	0.10102	2.42570	49	0.08346	2.10571	79	0.38081	1.91568
20	0.09983	2.42570	50	0.07201	2.09434	80	0.40350	1.88934
21	0.10027	2.42580	51	0.08560	2.08857	81	0.41583	1.86930
22	0.09961	2.42560	52	0.08417	2.06330	82	0.40538	1.87902
23	0.09977	2.42560	53	0.08956	2.05235	83	0.39238	1.87628
24	0.10076	2.42560	54	0.08040	2.04317	84	0.38860	1.89360
25	0.09957	2.42570	55	0.07032	2.04000	85	0.37935	1.90238
26	0.09971	2.42560	56	0.07955	2.03635	86	0.36738	1.91682
27	0.10009	2.42570	57	0.07576	2.02520	87	0.35601	1.92685
28	0.09937	2.42580	58	0.07385	2.01943	88	0.34636	1.93913
29	0.10019	2.42570	59	0.07942	2.00156	89	0.34632	1.94537
30	0.10039	2.42570	60	0.07956	1.99837	90	0.33539	1.95384