Supplementary Material - Speeding up the Hyperparameter Optimization of Deep Convolutional Neural Networks

Tobias, Nicolás Navarro-Guerrero, Sven Magg, and Stefan Wermter

Knowledge Technology, Department of Informatics,
Universität Hamburg
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
www.knowledge-technology.info

August 15, 2017

Abstract

This is the supplementary material to the paper “Speeding up the Hyperparameter Optimization of Deep Convolutional Neural Networks”. Here we give more details and additional results for our experiments. We will first describe the implementation details of the genetic algorithm (GA), followed by the hyperparameter search spaces used for the different optimization algorithms in the first experiment. Then we will give an overview over the best hyperparameters for the different data sets and resolutions, as found by the various optimization algorithms during the first experiment. Following this, we will give an overview of the importance of various subsets of hyperparameters for the STL-10 data set [7], similarly as in the original paper for the CK+ data set [10]. Finally, we give an overview over the hyperparameter search spaces and the best hyperparameters found in the second experiment on the 102-Flowers data set [11].

1 Implementation of the Genetic Algorithm

The general process of the GA is depicted in Figure 1. First, a population of convolutional neural networks (CNNs) is generated with random hyperparameters. After evaluating each CNN’s fitness, some CNNs are selected as “parents”. These parents are recombined to create “children” and these children together with the three best CNNs from the previous generation constitute the new generation. Finally, individual hyperparameters can mutate to increase the variance of hyperparameter values. Each optimization run is executed for a total of 30 generations.

We also measure the average time it takes to evaluate one generation. After each generation the three best CNNs are automatically transferred to the next generation without being evaluated again. Except for the very first generation, one generation, therefore, corresponds to 47 evaluations of CNNs. As a consequence, we have 50 evaluations in the first generation and 47 evaluations in the following 29 generations, for a total of $50+29*47 = 1413$ evaluations. We will now describe the general implementation of the GA.

Initialization For each optimization procedure with our GA, we initialize a population of 50 CNNs. The CNNs are encoded with a direct encoding strategy for which the hyperparameters that are to be optimized are encoded as genes, while all other hyperparameters, such as the activation function, stay the same. The hyperparameters are initially drawn from a given range. However, through mutation, the hyperparameters can take values outside the original range during the optimization process. Table 1 presents the list of optimized and fixed hyperparameters used in each of our experiments.

We also adhere to common architectures and guidelines and ensure in a second step that a higher convolutional layer does not have fewer filters than the previous convolutional layer. Analogous to how we handle the filters in the convolutional layers we ensure that higher hidden layers do not have more hidden units than lower hidden layers. This is some domain knowledge for CNNs that is inserted into the optimization process to speed up the time until good results are found. See Figure 2 for an overview of the optimized hyperparameters.

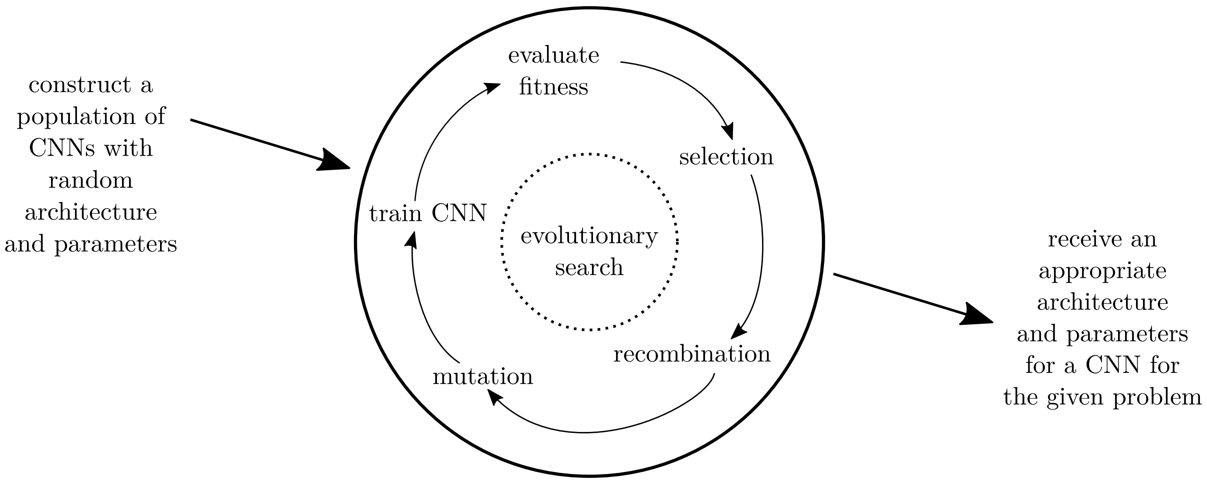
construct a population of CNNs with random architecture and parameters
evaluate fitness
selection
recombination
mutation
train CNN
evolutionary search
receive an appropriate architecture and parameters for a CNN for the given problem

Figure 1: General procedure of the hyperparameter optimization algorithm.

Table 1: Overview of fixed and optimized hyperparameters for the GA during the experiments.

	Hyperparameters	Values
Fixed	Activation Function	Rectified Linear Unit
	Pooling Operation	Max Pooling
	Pool Size	(2,2)
	Weight Initialization	$\mathcal{N}(0, \frac{2}{n})$[1]
	Momentum	0.9
	Early Stopping	5 Epochs w/o Improvement
	Max. Number of Epochs	50
Optimized	Filter Size	$\{(x,x) \mid (x > 2) \wedge (x \,\%\, 2 = 1)\}$
	Batch Size	$\{x \mid (x > 0) \wedge (x \,\%\, 10 = 0)\}$
	L1 Penalty	$\{0 \vee 10^{-x} \mid x \in \mathbb{N}_{>0}\}$
	L2 Penalty	$\{0 \vee 10^{-x} \mid x \in \mathbb{N}_{>0}\}$
	Learning Rate	$\{10^{-x} \mid x \in \mathbb{N}_{>0}\}$
	Number of Conv Layers	$\{x \mid x \in \mathbb{N}_{>0}\}$
	Filters per Conv Layer	$\{x \mid (x \in \mathbb{N}_{>0}) \wedge (x \,\%\, 10 = 0)\}$
	Number of Hidden Layers	$\{x \mid x \in \mathbb{N}_{>0}\}$
	Units per Hidden Layer	$\{x \mid (x \in \mathbb{N}_{>0}) \wedge (x \,\%\, 50 = 0)\}$

[1] n: number of incoming connections to one unit, see [8]

Training and Fitness Evaluation Each CNN is trained using stochastic gradient descent for a maximum of 50 epochs. To shorten the overall training time we employ an early stopping strategy and stop training if the validation error does not decrease for five consecutive epochs of training. The CNN's fitness is its error on the validation set after training is completed. Since we want the validation error to be as small as possible, the overall goal is to minimize the CNNs' fitness values over time.

Parent Selection and Recombination To create the next generation of CNNs from the current generation we combine multiple methods. First, the best three CNNs are automatically added to the next generation, this technique is called *elitism*. Next, three randomly created "new" CNNs are also added to the next generation of individuals. Then, we randomly choose 44 parents from the current generation using tournament selection, by picking two random individuals from the current generation and ranking them according to their fitness. We then choose the better individual with a probability of 75%, while selecting the worse individual in 25% of the cases and add it to the pool of parents. These 44 parents are recombined to create 44 children, which fill up the pool of 50 individuals for the next generation. The recombination of two distinct parents is done via 1-point crossover. At the crossover point in the genome, the two parents are split up and the respective parts are recombined to form two new children. In order to retain some architectural features of a CNN we merge some individual hyperparameters during the recombination process. More precisely, all convolutional layers with associated filters and filter sizes,

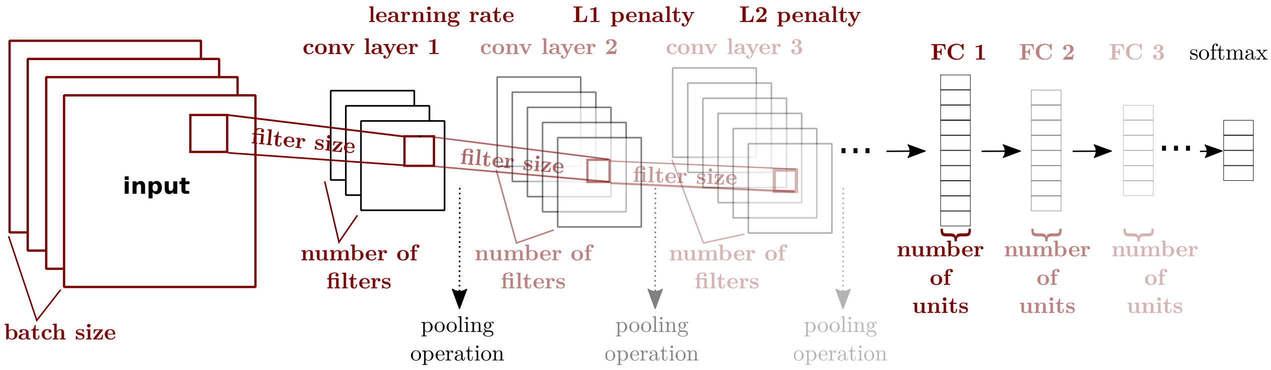

Figure 2: Overview of the hyperparameters that are optimized during the experiments.

as well as all hidden layers with associated units are treated as individual hyperparameters. See Figure 3 for a visualization of the recombination procedure.

Mutation After the recombination step, each individual's gene is mutated with a probability of 10%. Through this, the hyperparameters can take values that were not originally present. After the mutation step, it is again ensured that higher convolutional layers do not have fewer filters than lower layers and that higher hidden layers do not have more units than lower hidden layers.

2 Hyperparameter Search Space (Experiment 1)

Here, we provide information about the search and initialization space of the genetic algorithm, Random Search and the TPE algorithm in the first experiment. Similar to the genetic algorithm both Random Search and TPE were given a total of 1500 evaluations.

2.1 Genetic Algorithm

Initialization Tables 2 and 3 show the initialization parameters for the GA for the two data sets used in the first experiment. We, start with a relatively small number of filters per convolutional layer which can be increased over time by the GA. Additionally, a max-pooling layer is inserted after each convolutional layer. The values for many of the hyperparameters are adapted from both, commonly seen value ranges for CNNs on these data sets [9], and general recommendations [6, 2, 12]. If only a single value for the number of filters per layer is given, this is the maximum value *max* of filters for the respective convolutional layer, i.e. the number of filters for this layer is drawn from [10, *max*]

Table 2: Initialization of the hyperparameters for the CK+ data set in the first experiment.

hyperparameter	GA	RS and TPE
learning rate	{1E-1, 1E-2, 1E-3}	[0.00001, 0.1]
L1 penalty	{0.0, 1E-1, 1E-2, 1E-3, , 1E-4}	[0.000001, 0.1]
L2 penalty	{0.0, 1E-1, 1E-2, 1E-3, , 1E-4}	[0.000001, 0.1]
# conv layers (32/64/128/200px)	{1, 2} / {1, 2, 3} / {1, ..., 4} / {1, ..., 5}	
# filters (32px)	50, 50	150, 260
# filters (64px)	50, 50, 50	150, 260, 400
# filters (128px)	50, 50, 50, 50	150, 260, 400, 600
# filters (200px)	50, 50, 50, 50, 50	50, 100, 200, 300, 400
filter size	{3x3, 5x5, 7x7}	
# hidden layers	{1, 2, 3}	
# units per layer	[50, 500]	
batch size	{10, 20, 30, 40, 50}	

Table 3: Initialization of the hyperparameters for the STL-10 data set in the first experiment.

hyperparameter	GA	RS and TPE
learning rate	{1E-1, 1E-2, 1E-3}	[0.00001, 0.1]
L1 penalty	{0.0, 1E-1, 1E-2, 1E-3, , 1E-4}	[E-7, 0.1]
L2 penalty	{0.0, 1E-1, 1E-2, 1E-3, , 1E-4}	[E-7, 0.1]
# conv layers (32/48/96px)	{1, 2} / {1, 2, 3} / {1, 2, 3, 4}	
# filters (32px)	50, 50	200, 400
# filters (48px)	50, 50, 50	200, 400, 600
# filters (96px)	50, 50, 50, 50	200, 400, 600, 800
filter size	{3x3, 5x5, 7x7}	
# hidden layers	{1, 2, 3}	
# units per layer	[50, 500]	
batch size	{10, 20, 30, 40, 50, 60, 70, 80}	

Recombination Figure 3 illustrates an example including only three hyperparameters: the learning rate, the convolutional layers and the hidden layers. All other individual hyperparameters, such as the regularization penalties, are treated identically and can come from either parent, depending on the location of the crossover point.

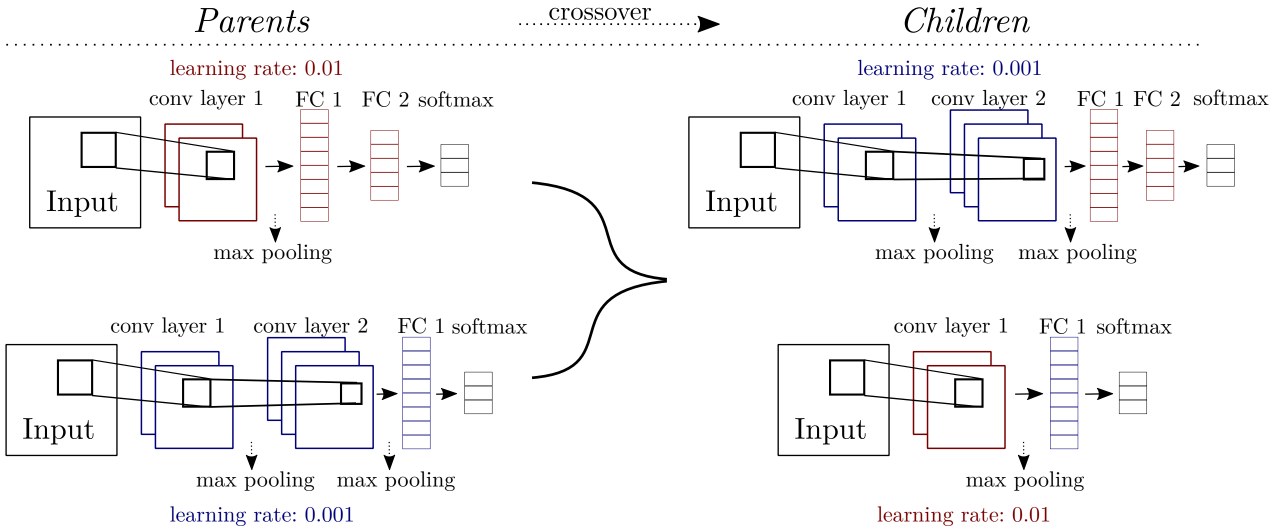

Figure 3: The two CNNs on the left are the parents with according architecture and learning rate. Each parent has one or two convolutional layers, one or two hidden, fully connected layers (FC) and one Softmax layer for the classification. After performing the crossover operation we obtain the two children on the right side.

Mutation Table 4 gives an overview over the mutation probabilities for the different hyperparameters. We have a slightly higher probability for decreasing the learning rate because we can already sample the biggest allowed learning rate of 0.1 from the very beginning. The tendency to decrease the learning rate is, therefore, advantageous if smaller learning rates need to be found. If they are mutated, the number of the convolutional or hidden layers is in- or decreased by one with a probability of 50% each.

The L1 and L2 regularization parameters are mutated in the same manner as the learning rate, but need some special treatment in case they become zero. If the penalty is zero, there is a 40% chance that it will be increased. If this is the case it is either set to 0.001 or 0.0001, each with a probability of 50%. If one of the penalties becomes smaller than 1E-5, it is set to zero. The mutation parameters are set manually at the beginning of the optimization process. They are not fine-tuned for the specific optimization task, but rather represent intuitive knowledge about how hyperparameters are usually adapted during manual or grid search. The exact value of the mutation parameters does most likely not have an impact on the final solution, but may have an impact on how quickly the algorithm converges.

Table 4: Mutation parameters for the first experiment. The columns describe the probability of the state changes of a given hyperparameter in case it is mutated.

hyperparameter	10%	30%	50%	hyperparameter	10%	40%
learning rate L1 penalty L2 penalty	$\times 10^2$ / $\times 10^{-2}$	$\times 10$	$\times 10^{-1}$	# filters	± 20	± 10
				# hidden units	± 100	± 50
				filter size	$\pm(4 \times 4)$	$\pm(2 \times 2)$
				batch size	± 20	± 10

2.2 Random Search

For the implementation of the random search algorithm, we use the Hyperopt library[1] [4] to randomly sample hyperparameters from a previously defined search space. See Tables 2 and 3 for details on the search space for the individual hyperparameters and data sets.

2.3 TPE

The "Tree of Parzen Estimators" (TPE) algorithm [3] is a sequential model-based optimization (SMBO) strategy based on Bayesian probabilities. We use the implementation of TPE provided by Hyperopt[1] [4]. It takes as input a specification of the initial search space for the hyperparameters, a loss function, which maps specific hyperparameters to a real value and an experimental history of values of the loss function [5]. The algorithm then returns a suggestion for which hyperparameter setting should be evaluated next, given the current search history. The next hyperparameter setting is then evaluated using the loss function, added to the search history and the next suggestion is calculated based on the updated history. Again, the five best performing hyperparameter settings are kept for further analysis and comparison. The definition of the search space is the same as for the random search algorithm, see Tables 2 and 3.

---

[1] available from `https://github.com/hyperopt/hyperopt`

3 Best Hyperparameters per Algorithm (Experiment 1)

Here we provide a detailed overview of the hyperparameters that were found by the various optimization algorithms for the different data sets in the first experiment. Tables 5, 6 and 7 describe the hyperparameters found for the CK+ data set, while Tables 8, 9 and 10 describe the hyperparameters for the STL-10 data set and Table 11 describes the hyperparameters used during the evaluation on the test sets.

Table 5: Results of the best CNNs found by the **genetic algorithm** for the extended **Cohn-Kanade** data set. The table shows results averaged over three independent optimization runs for each resolution. For each of the four resolutions the genetic algorithm was initialized randomly from the same search space. The crossover and mutation parameters stay the same across all resolutions.

input size	32x32px	64x64px
avg generation time	5 min 06 sec	7 min 33 sec
avg validation error	0.048 ± 0.007	0.043 ± 0.011
learning rate	0.01 (100%)	0.1 (13%), 0.01 (87%)
batch size	10 (100%)	10 (80%), 50 (13%)
L1 regularization	0.1 (27%), 0.01 – 0.001 (47%)	0.01 – 0.1 (33%), 0.0001 (53%)
L2 regularization	0 (60%), 0.01 – 0.1 (20%)	0 (13%) 0.01 – 0.1 (87%)
number of conv layers	1 (73%), 2 (27%)	2 (100%)
number of hidden layers	1 (100%)	1 (100%)
number of filters		
1st conv layer	20 – 30 (47%), 50 – 80 (53%)	10 (100%)
2nd conv layer		20 – 40 (100%)
filter size		
1st conv layer	5 (80%), 9 (20%)	5 (100%)
2nd conv layer		3 (67%), 5 (33%)
units in 1st hidden layer	200 – 300 (47%), 400 – 450 (47%)	200 – 250 (67%), 350 – 400 (33%)

input size	128x128px	200x200px
avg generation time	14 min 05 sec	20 min 44 sec
avg validation error	0.047 ± 0.011	0.058 ± 0.013
learning rate	0.01 (100%)	0.01 (100%)
batch size	10 (100%)	10 (73%), 40 (13%)
L1 regularization	0.001 (20%), 0.01 – 0.1 (73%)	0 – 0.001 (60%), 0.01 – 0.1 (33%)
L2 regularization	0 (60%), 0.1 (40%)	0.001 (33%), 0.01 – 0.1 (60%)
number of conv layers	3 (40%), 4 (60%)	4 (67%), 5 (33%)
number of hidden layers	1 (100%)	1 (87%)
number of filters		
1st conv layer	10 (67%), 60 (33%)	10 – 20 (93%)
2nd conv layer	30 – 50 (67%), 90 (33%)	10 – 30 (87%)
3rd conv layer	60 – 80 (67%), 90 – 100 (33%)	40 – 70 (87%)
4th conv layer	100 (100%)	50 – 70 (47%), 80 – 100 (53%)
filter size		
1st conv layer	5 (67%), 9 (33%)	5 (67%), 7 (33%)
2nd conv layer	3 (67%), 5 (27%)	3 (67%), 5 (33%)
3rd conv layer	3 (93%)	3 (100%)
4th conv layer	3 (100%)	3 (100%)
units in 1st hidden layer	200 – 300 (27%), 350 – 450 (73%)	300 – 350 (27%), 400 – 500 (60%)

Table 6: Results of the best CNNs found by **Random Search** for the extended **Cohn-Kanade** data set. The table shows results averaged over three independent optimization runs for each resolution. For each optimization run of the four resolutions, random search sampled 1500 hyperparameter settings from the search space and evaluated their performance.

input size	32x32px	64x64px
avg generation time	5 min 26 sec	11 min 42 sec
avg validation error	0.064 ± 0.007	0.078 ± 0.005
learning rate	0.008 – 0.08	0.001 – 0.04
batch size	10 – 20 (53%), 30 – 40 (47%)	10 – 20 (53%), 30 – 40 (40%)
L1 regularization	0.0 (47%), 0.001 – 0.01 (33%)	0.0 (60%), 0.002 – 0.02 (33%)
L2 regularization	0.0 (33%), 0.01 – 0.1 (40%)	0.0 (33%), 0.001 – 0.05 (47%)
# conv layers	1 (80%), 2 (20%)	2 (27%), 3 (73%)
# hidden layers	1 (67%), 2 (20%)	1 (80%), 2 (20%)
number of filters		
1st conv layer	30 – 80 (60%), 90 – 120 (27%)	30 – 70 (60%), 100 – 120 (27%)
2nd conv layer		20 – 60 (40%), 70 – 120 (33%)
3rd conv layer		100 – 200 (45%), 210 – 300 (36%)
filter size		
1st conv layer	5 (33%), 7 (60%)	5 (47%), 7 (40%)
2nd conv layer		3 (20%), 5 (47%), 7 (33%)
3rd conv layer		3 (27%), 5 (18%), 7 (55%)
units in 1st hidden layer	100 – 300 (33%), 400 – 500 (47%)	100 – 300 (60%), 400 – 500 (40%)

input size	128x128px	200x200px
avg generation time	37 min 13 sec	42 min 42 sec
avg validation error	0.07 ± 0.007	0.07 ± 0.01
learning rate	0.003 – 0.05	0.003 – 0.06
batch size	10 – 20 (40%), 30 – 40 (47%)	10 – 20 (40%), 30 – 40 (47%)
L1 regularization	0.0 (40%), 0.0001 – 0.01 (47%)	0.0 (53%), 0.001 – 0.01 (33%)
L2 regularization	0.01 – 0.1 (40%) 0.0001 – 0.009 (40%)	0.001 – 0.01 (27%) 0.01 – 0.1 (47%)
# conv layers	3 (73%), 4 (27%)	4 (73%), 5 (27%)
# hidden layers	1 (73%), 2 (27%)	1 (80%), 2 (20%)
number of filters		
1st conv layer	20 – 80 (87%)	10 – 50 (100%)
2nd conv layer	100 – 150 (47%), 170 – 230 (47%)	20 – 70 (33%), 80 – 130 (53%)
3rd conv layer	200 – 270 (40%), 300 – 360 (33%)	50 – 100 (27%), 120 – 200 (60%)
4th conv layer		150 – 300 (73%)
filter size		
1st conv layer	3 (27%), 5 (27%), 7 (46%)	5 (20%), 7 (73%)
2nd conv layer	3 (47%), 5 (40%), 7 (13%)	5 (47%), 7 (40%)
3rd conv layer	3 (33%), 5 (40%), 7 (17%)	3 (20%), 5 (53%), 7 (27%)
4th conv layer		3 (20%), 5 (33%), 7 (47%)
units in 1st hidden layer	100 – 250 (80%)	100 – 250 (53%), 300 – 450 (27%)

Table 7: Results of the best CNNs found by the **TPE algorithm** for the extended **Cohn-Kanade** data set. The table shows results averaged over three independent optimization runs for each resolution. For each of the four resolutions, the TPE algorithm employed the same search space. Similar to random search the TPE algorithm got 1500 evaluations per optimization run and resolution.

input size	32x32px	64x64px
avg generation time	7 min 03 sec	12 min 10 sec
avg validation error	0.046 ± 0.007	0.046 ± 0.01
learning rate	0.006 − 0.009	0.01 − 0.05
batch size	10 (60%), 20 (20%)	10 (47%), 40 (33%)
L1 regularization	0.0 (47%), 0.003 − 0.03 (27%)	0.0 (47%), 0.0001 − 0.0005 (40%)
L2 regularization	0.005 − 0.05 (80%)	0.0 (40%), 0.02 − 0.06 (33%)
number of conv layers	1 (67%), 2 (33%)	2 (67%), 3 (33%)
number of hidden layers	1 (67%), 2 (33%)	1 (93%)
number of filters		
1st conv layer	30 − 60 (53%), 100 − 130 (40%)	20 − 40 (93%)
2nd conv layer	150 − 180 (100%)	100 − 150 (47%), 160 − 180 (33%)
filter size		
1st conv layer	5 (47%), 7 (53%)	5 (87%), 7 (13%)
2nd conv layer	5 (100%)	5 (53%), 7 (27%)
units in 1st hidden layer	250 − 350 (60%), 400 − 500 (27%)	100 − 200 (47%), 300 − 350 (40%)

input size	128x128px	200x200px
avg generation time	51 min 18 sec	59 min 07 sec
avg validation error	0.051 ± 0.008	0.041 ± 0.01
learning rate	0.007 − 0.04	0.01 − 0.05
batch size	30 (47%), 40 (33%)	10 (33%), 30 (60%)
L1 regularization	0.0 (53%), 0.002 − 0.02 (40%)	0.0 (60%), 0.01 − 0.1 (33%)
L2 regularization	0.0 (27%), 0.02 − 0.09 (53%)	0.0 (27%), 0.01 − 0.1 (53%)
number of conv layers	3 (40%), 4 (60%)	4 (67%), 5 (33%)
number of hidden layers	1 (100%)	1 (80%)
number of filters		
1st conv layer	10 − 20 (100%)	10 − 20 (53%), 30 − 50 (47%)
2nd conv layer	10 − 30 (47%), 40 − 70 (40%)	110 − 150 (87%)
3rd conv layer	120 − 170 (67%), 200 − 260 (27%)	80 − 120 (67%), 200 − 240 (27%)
4th conv layer	470 − 570 (56%)	200 − 270 (27%), 330 − 390 (60%)
filter size		
1st conv layer	5 (60%), 7 (33%)	5 (67%), 7 (33%)
2nd conv layer	3 (27%), 7 (53%)	3 (53%), 5 (33%)
3rd conv layer	3 (53%), 7 (33%)	3 (33%), 7 (40%)
4th conv layer	3 (44%), 7 (33%)	5 (33%), 7 (40%)
units in 1st hidden layer	100 − 200 (53%), 250 − 300 (33%)	50 − 150 (47%), 400 − 500 (47%)

Table 8: Results of the best CNNs found by the **genetic algorithm** for the **STL-10** data set. The table shows results averaged over three independent optimization runs for each resolution. For each of the three resolutions, the genetic algorithm was initialized randomly from the same search space. The crossover and mutation parameters stay the same across all resolutions.

input size	32x32px	48x48px	96x96px
avg generation time	13 min 07 sec	17 min 56 sec	29 min 56 sec
avg validation error	0.580 ± 0.001	0.579 ± 0.001	0.539 ± 0.002
learning rate	0.001 (100%)	0.001 (100%)	0.001 (100%)
batch size	20 (20%), 30 (20%) 40 (27%), 50 (33%)	20 (53%), 30 (13%) 40 (13%), 50 (13%)	10 (20%), 30 (27%) 40 (27%), 50 (20%)
L1 regularization	0.0 (47%) 0.001 – 0.01 (47%)	0.001 – 0.01 (47%) 0.1 (47%)	0.0 (67%) 0.01 (33%)
L2 regularization	0.0 (33%) 0.1 (67%)	0.001 – 0.01 (53%) 0.1 (40%)	0.0 (33%) 0.01 – 0.1 (53%)
# conv layers	2 (100%)	3 (100%)	4 (100%)
# hidden layers	2 (100%)	2 (100%)	2 (100%)
# filters			
1st conv layer	40 (93%)	20 – 30 (67%) 40 – 50 (33%)	40 – 50 (100%)
2nd conv layer	50 (93%)	50 – 60 (73%) 70 – 80 (27%)	30 (33%) 60 – 70 (67%)
3rd conv layer		60 – 70 (47%)	60 – 80 (80%)
4th conv layer		80 – 90 (53%)	100 (80%)
filter size			
1st conv layer	3 (100%)	3 (100%)	3 (67%), 5 (33%)
2nd conv layer	3 (100%)	3 (100%)	3 (100%)
3rd conv layer		3 (100%)	3 (100%)
4th conv layer			3 (100%)
# hidden units			
1st hidden layer	350 – 400 (47%) 450 – 500 (53%)	400 – 450 (47%) 500 – 600 (47%)	300 – 350 (87%)
2nd hidden layer	100 (33%), 200 (33%) 300 (33%)	100 – 200 (53%) 450 – 500 (47%)	200 (60%) 350 (33%)

Table 9: Results of the best CNNs found by **Random Search** for the **STL-10** data set. The table shows results averaged over three independent optimization runs for each resolution. For each optimization run of the three resolutions, 1500 hyperparameter settings were randomly sampled from the search space and their performance was evaluated.

input size	32x32px	48x48px	96x96px
avg generation time	13 min 22 sec	27 min 54 sec	1h 07 min 42 sec
avg validation error	0.594 ± 0.005	0.58 ± 0.013	0.573 ± 0.008
learning rate	0.0001 − 0.003	0.0002 − 0.001	0.0001 − 0.001
batch size	10 (20%), 20 (20%) 40 (27%), 70 (20%)	10 (20%), 20 (33%) 30 (33%)	10 (30%), 20 (20%) 50 (40%)
L1 regularization	0.0 (47%) 0.001 − 0.01 (40%)	0.0 (33%) 0.001 − 0.05 (40%)	0.0 (40%) 0.001 − 0.05 (50%)
L2 regularization	0.0 (53%) 0.01 − 0.02 (33%)	0.0001 − 0.001 (40%) 0.01 − 0.1 (27%)	0.0 (50%) 0.001 − 0.06 (50%)
# conv layers	2 (100%)	3 (93%)	3 (20%), 4 (80%)
# hidden layers	2 (60%), 3 (40%)	2 (80%), 3 (20%)	2 (70%), 3 (20%)
# filters			
1st conv layer	50 − 70 (33%)	30 − 50 (40%)	50 − 80 (40%)
	80 − 110 (40%)	140 − 160 (33%)	100 − 150 (50%)
2nd conv layer	100 − 190 (27%)	100 − 200 (33%)	150 − 200 (60%)
	200 − 300 (53%)	210 − 300 (40%)	210 − 250 (20%)
3rd conv layer		200 − 300 (33%)	300 − 500 (60%)
4th conv layer		400 − 600 (42%)	500 − 800 (75%)
filter size			
1st conv layer	3 (80%), 5 (20%)	3 (80%), 5 (13%)	3 (40%), 5 (40%)
2nd conv layer	3 (67%), 5 (33%)	3 (80%), 7 (14%)	3 (40%), 5 (40%)
3rd conv layer		3 (57%), 5 (21%)	5 (60%), 7 (20%)
4th conv layer			3 (63%), 5 (25%)
# hidden units			
1st hidden layer	250 − 350 (40%)	250 − 350 (40%)	100 − 250 (50%)
	400 − 500 (53%)	400 − 450 (47%)	350 − 450 (50%)
2nd hidden layer	100 − 150 (20%)	200 − 300 (33%)	150 − 250 (67%)
	200 − 350 (67%)	350 − 450 (53%)	400 − 500 (22%)

Table 10: Results of the best CNNs found by the **TPE algorithm** for the **STL-10** data set. The table shows results averaged over three independent optimization runs for each resolution. For each of the three resolutions, the TPE algorithm employed the same search space. Similar to random search the TPE algorithm got 1500 evaluations per optimization run and resolution.

input size	32x32px	48x48px	96x96px
avg generation time	18 min 52 sec	31 min 33 sec	2h 41 min 33 sec
avg validation error	0.577 ± 0.002	0.553 ± 0.004	0.533 ± 0.003
learning rate	0.0003 − 0.0005	0.0002 − 0.0008	0.0003 − 0.0008
batch size	10 (20%), 20 (80%)	10 (100%)	40 (100%)
L1 regularization	0.0 (100%)	0.0004 − 0.0009 (80%)	0.0 (100%)
L2 regularization	0.09 − 0.13 (80%)	0.04 − 0.1 (100%)	0.001 − 0.009 (100%)
# conv layers	2 (100%)	3 (100%)	4 (100%)
# hidden layers	3 (100%)	2 (100%)	2 (100%)
# filters			
1st conv layer	150 − 200 (100%)	90 − 110 (100%)	180 − 200 (100%)
2nd conv layer	340 − 390 (100%)	190 − 210 (80%)	50 − 90 (40%) 140 − 150 (60%)
3rd conv layer		400 − 420 (60%)	360 − 450 (80%)
4th conv layer			570 − 690 (100%)
filter size			
1st conv layer	3 (100%)	3 (100%)	3 (100%)
2nd conv layer	3 (100%)	3 (100%)	3 (100%)
3rd conv layer		3 (80%)	3 (100%)
4th conv layer			5 (60%), 7 (40%)
# hidden units			
1st hidden layer	450 − 500 (100%)	450 − 500 (100%)	450 − 500 (80%)
2nd hidden layer	200 − 250 (80%)	400 − 450 (100%)	450 − 500 (80%)

Table 11: Results of **10-fold cross-validation** on the extended **Cohn-Kanade** and the **STL-10** data set. Each setting was tested both without any additional regularization and with regularization methods such as dropout and batch normalization (bn). The best results are highlighted in bold.

Data set	**Found by**	**Hyperparameters**	**Regularization**	**Val error**

Extended Cohn Kanade	Genetic Algorithm	learning rate = 0.01, batch size = 10
		# conv layers = 3, filter size = [5,3,3]
		filters = [10,40,60]
		# hidden layers = 1, units = [350]
		L1 = 0.0001, L2 = 0.1
			none	0.057 ± 0.005

			dropout	**0.046 ± 0.003**
			dropout max dropout	0.716 ± 0.011

		learning rate = 0.01, batch size = 10
		# conv layers = 4, filter size = [5,3,3,3]
		filters = [10,40,80,100]
		# hidden layers = 1, units = [300]
		L1 = 0.01, L2 = 0.1
			none	0.057 ± 0.005

			dropout	**0.046 ± 0.002**
			dropout max dropout	0.627 ± 0.065

	Random Search	learning rate = 0.018, batch size = 20
		# conv layers = 3, filter size = [7,5,3]
		filters = [50,200,320]
		# hidden layers = 1, units = [500]
		L1 = 0.0, L2 = 0.1
			none	0.077 ± 0.041

			dropout	**0.043 ± 0.004**
			dropout max dropout	0.813 ± 0.000

	TPE	learning rate = 0.029, batch size = 30
		# conv layers = 3, filter size = [7,7,3]
		filters = [70,150,340]
		# hidden layers = 1, units = [150]
		L1 = 0.013, L2 = 0.02
			none	0.062 ± 0.001

			dropout	0.066 ± 0.007
			dropout **bn**	**0.033 ± 0.001**

		learning rate = 0.032, batch size = 30
		# conv layers = 4, filter size = [5,7,3,3]
		filters = [10,140,220,470]
		# hidden layers = 1, units = [200]
		L1 = 0.0, L2 = 0.0001
			none	0.079 ± 0.002

			dropout	**0.053 ± 0.003**
			dropout bn	0.06 ± 0.003

STL-10	Genetic Algorithm	learning rate = 0.001, batch size = 40
		# conv layers = 4, filter size = [5,3,3,3]
		filters = [50,70,100,140]
		# hidden layers = 2, units = [350,350]
		L1 = 0.0, L2 = 0.01
			none	0.544 ± 0.001

			dropout	0.527 ± 0.001
			dropout + bn	0.862 ± 0.003
			dropout + data augm	**0.467 ± 0.003**

	Random Search	learning rate = 0.00087, batch size = 50
		# conv layers = 4, filter size = [7,5,5,5]
		filters = [110,170,400,330]
		# hidden layers = 2, units = [250,450]
		L1 = 0.0, L2 = 0.006
			none	0.569 ± 0.001

			dropout	0.556 ± 0.015
			dropout + bn	0.579 ± 0.001
			dropout + data augm	**0.482 ± 0.001**

	TPE	learning rate = 0.00087, batch size = 40
		# conv layers = 4, filter size = [5,3,3,3]
		filters = [200,140,380,690]
		# hidden layers = 2, units = [400,450]
		L1 = 0.0, L2 = 0.0155
			none	0.529 ± 0.001

			dropout	0.579 ± 0.049
			dropout + bn	0.541 ± 0.012
			data augmentation	**0.473 ± 0.001**

4 Meta-Analysis of GA, TPE, and Hyperparameters

Here, we present visualizations of the impact of various hyperparameters on the STL-10 data set, similar to the results shown in the original paper for the CK+ data set. Figure 4 presents an overview of the progress of the optimization procedures. Figures 5, 6 and 7 show the importance of various subsets of hyperparameters, the impact of the learning rate and the impact of the learning rate together with the number of hidden layers for the STL-10 data set. For the visualization of the importance and effects of the hyperparameters we again aggregated the data from all runs of all optimization algorithms.

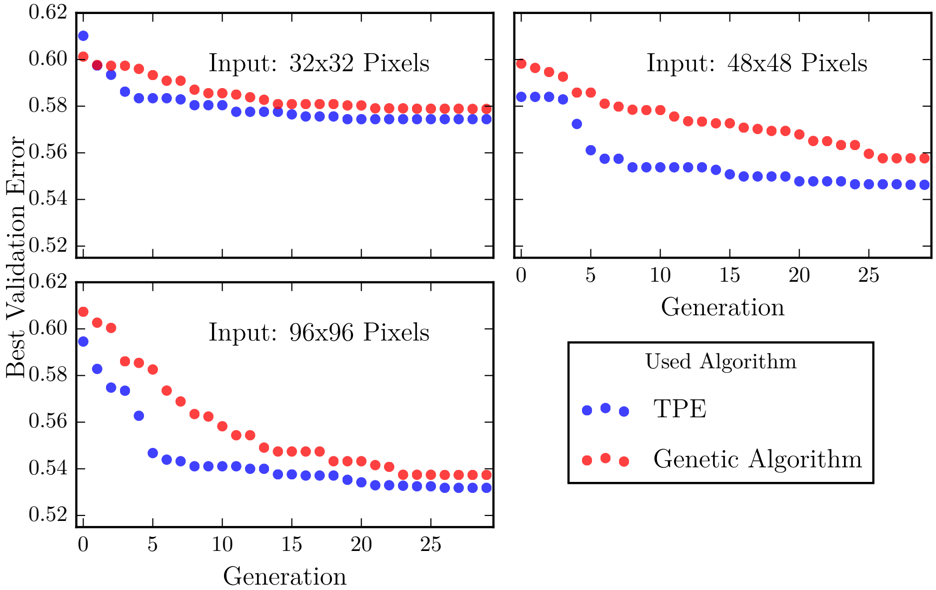

Figure 4: Best average validation error after each generation for the genetic algorithm and the TPE algorithm on the STL-10 data set over a total of 30 generations.

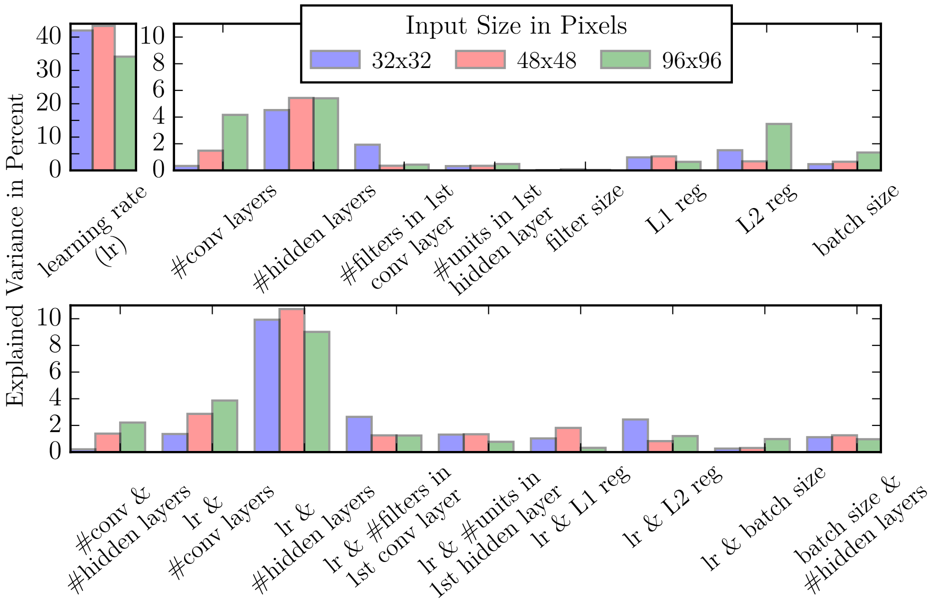

Figure 5: Explained variance of the validation error in percent on the STL-10 data set, based on the results of all optimization algorithms.

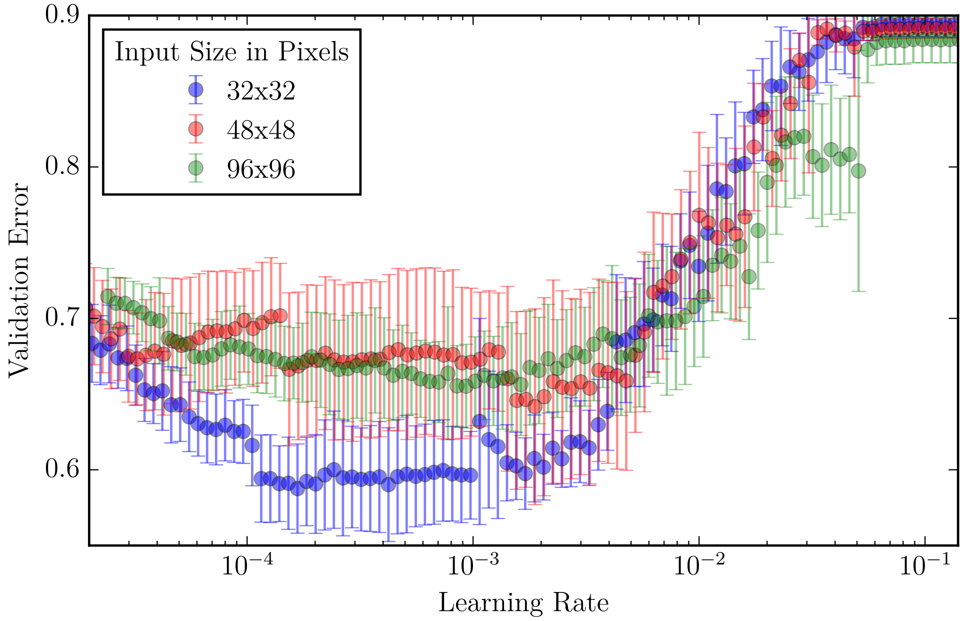

Figure 6: Merging the results of all three optimization algorithms shows the relationship between the learning rate and the validation error for different image resolutions on the STL-10 data set. We can see that the best learning rate is similar for all resolutions, usually between 10^{-3} and 10^{-4}.

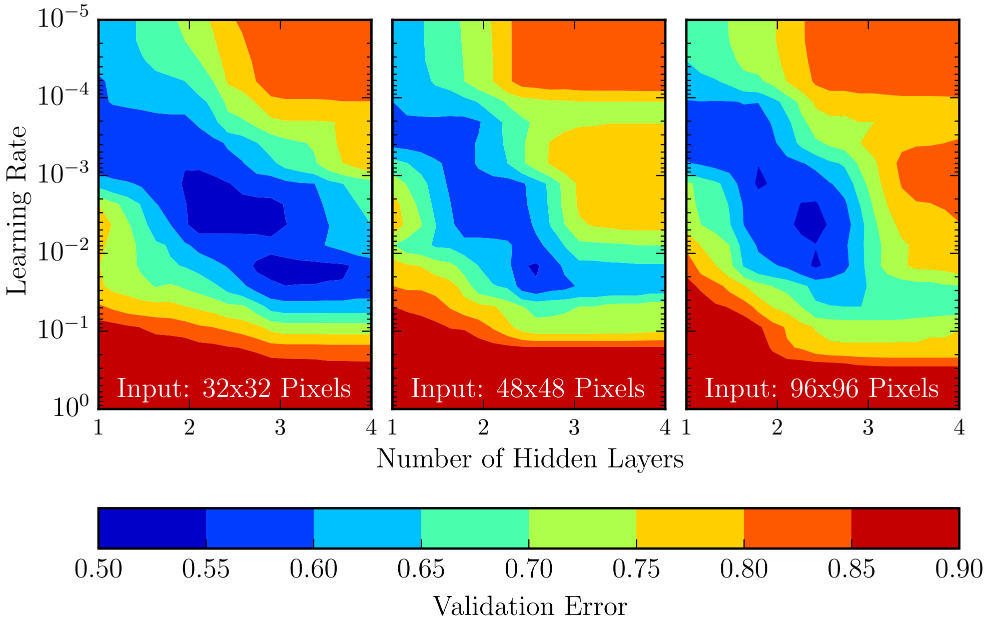

Figure 7: Merging the results of all three optimization algorithms shows the relationship between the learning rate, the number of hidden layers and the validation error for different image resolutions on the STL-10 data set.

5 Hyperparameter Search Space (Experiment 2)

Several researchers have achieved good results on the 102 Flowers data set using common CNNs with many layers and filters [13, 1]. These CNNs are based on some of the winners of ImageNet Competitions of the last years, but due to hardware limitations, we are not able to use them directly. However, their hyperparameters can give us some suggestions for an initial search space. Since the 102 Flowers data set contains many more classes than the previous data sets we tested and the exemplary CNNs that achieve good results are very big, we change the initialization for the GA. We now describe the initializations for the GA optimization runs on the different input sizes, as well as for TPE, Random Search, and SMAC[2]. The initialization and handling of hyperparameters that are not mentioned explicitly are the same as in the first experiment.

For the initial search space we use the hyperparameters of Sermanet et al. [12], Razavian et al. [13] and Azizpour et al. [1] as coarse direction, but reduce the number of layers, filters and hidden units due to hardware constraints. The initializations that are made for higher input sizes for the optimization algorithms are based on the results of the previous, smaller input size. The initialization and mutation parameters were then adapted to reduce the variance in the respective hyperparameters. The idea behind this is that the optimization process on the smaller data sizes should already choose "good" areas in the hyperparameter space which should then be examined in more detail. For more details on the exact mutation parameters that are used for the different resolutions see Table 12. Table 13 shows the initialization of the hyperparameters for the various resolutions of the GA, while Tables 14, 15 and 16 show the search space for Random Search, TPE and SMAC for the various resolutions on the 102 Flowers data set.

Table 12: The first horizontal block describes the mutation parameters for all hyperparameters for the smallest resolution of 32x32 pixels. The second and third horizontal blocks describe the mutation parameters that changed relative to the previous, smaller resolution.

32x32px	10%	40%	50%
# conv layers # hidden layers			±1
learning rate L1 penalty L2 penalty	$\times 10^2$ / $\times 10^{-2}$	$\times 10$ / $\times 10^{-1}$	
# filters	$+30/+50$	±20	
# hidden units	±100	±50	
filter size			$\pm(2 \times 2)$
batch size	±30	±10	
64x64px	**10%**	**40%**	**50%**
learning rate	$\times 10$ / $\times 10^{-1}$	$\times 5$ / $\times 5^{-1}$	
# filters	±10	±20	
# hidden units	±100	±50	
128x128px	**10%**	**40%**	**50%**
learning rate	$\times 5$ / $\times 5^{-1}$	$\times 2$ / $\times 2^{-1}$	
batch size			±10

---

[2] For our experiments we used SMAC version 2.10.03. For further information see `http://www.cs.ubc.ca/labs/beta/Projects/SMAC/`

Table 13: **Initialization** of the hyperparameters of the **GA** for the 102 Flowers data set in the second experiment.

hyperparameter	32 (15gen)	64 (10gen)	128 (5gen)	128 (30gen)
learning rate	{0.001, 0.01, 0.1}	{0.01, 0.1}	{0.005, 0.01, 0.05, 0.1}	{0.001, 0.01, 0.1}
L1 penalty	[0.0, 0.1]	{0.0, 0.00001}	{0.0}	[0.0, 0.1]
L2 penalty	[0.0, 0.1]	[0.0, 0.1]	[0.0, 0.1]	[0.0, 0.1]
# conv layers	{1, 2}	{2, 3, 4}	{4}	{1, ..., 5}
# filters	50-300	100-200, 150-300	150-300, 200-400	10-700
	50-300	250-400, 300-450	250-450, 300-500	
filter size	{3x3, 5x5, 7x7}	{3x3, 5x5, 7x7}	{3x3, 5x5}	{3x3, 5x5, 7x7}
# hidden layers	{1, 2, 3}	{1, 2}	{1, 2}	{1, 2, 3}
# hidden units	[100, 1000]	[500-1000]	[500-1000]	[100, 1000]
batch size	[20, 100]	[20, 50]	[10, 40]	[10, 100]

Table 14: **Search space** of the hyperparameters of **Random Search** for the 102 Flowers data set in the second experiment.

hyperparameter	32 (15gen)	64 (10gen)	128 (5gen)	128 (30gen)
learning rate	[0.00001, 0.1]	[0.001, 0.1]	[0.005, 0.05]	[0.00001, 0.1]
L1 penalty	[0.0, 0.1]	[0.0, 0.00001]	[0.0, 0.00001]	[0.0, 0.1]
L2 penalty	[0.0, 0.1]	[0.0, 0.1]	[0.0, 0.1]	[0.0, 0.1]
# conv layers	{1, 2}	{3, 4}	{4, 5}	{1, ..., 5}
# filters	10-150	50-200, 100-400	50-180, 150-350, 350-550	10-150, 10-250, 10-350
	10-300	150-600, 200-800	400-700, 450-800	10-500, 10-600
filter size	{3x3, 5x5, 7x7}	{3x3, 5x5, 7x7}	{3x3, 5x5, 7x7}	{3x3, 5x5, 7x7}
# hidden layers	{1, 2, 3}	{1, 2}	{1, 2}	{1, 2, 3}
# hidden units	[100, 1000]	[300, 800]	[400, 800]	[100, 1000]
batch size	[10, 250]	[10, 150]	[10, 80]	[10, 150]

Table 15: **Search space** of the hyperparameters of **TPE** for the 102 Flowers data set in the second experiment.

hyperparameter	32 (15gen)	64 (10gen)	128 (5gen)	128 (30gen)
learning rate	[0.00001, 0.1]	[0.003, 0.1]	[0.005, 0.05]	[0.00001, 0.1]
L1 penalty	[0.0, 0.1]	[0.0, 0.0001]	[0.0, 0.00001]	[0.0, 0.1]
L2 penalty	[0.0, 0.1]	[0.0, 0.1]	[0.0, 0.001]	[0.0, 0.1]
# conv layers	{1, 2}	{3, 4}	{4, 5}	{1, ..., 5}
# filters	10-150	40-150, 150-300	80-150, 150-250, 250-350	10-150, 10-300, 10-500
	10-300	200-400, 250-500	400-500, 450-600	10-600, 10-700
filter size	{3x3, 5x5, 7x7}	{3x3, 5x5, 7x7}	{3x3, 5x5, 7x7}	{3x3, 5x5, 7x7}
# hidden layers	{1, 2, 3}	{1, 2}	{1, 2}	{1, 2, 3}
# hidden units	[100, 1000]	[400, 1200]	[800, 1300]	[100, 1000]
batch size	[10, 250]	[10, 120]	[10, 50]	[10, 150]

Table 16: **Search space** of the hyperparameters of **SMAC** for the 102 Flowers data set in the second experiment.

hyperparameter	32 (15gen)	64 (10gen)	128 (5gen)	128 (30gen)
learning rate	[0.00001, 0.1]	[0.001, 0.1]	[0.003, 0.04]	[0.00001, 0.1]
L1 penalty	[0.0, 0.1]	[0.0, 0.0001]	[0.0, 0.000001]	[0.0, 0.1]
L2 penalty	[0.0, 0.1]	[0.001, 0.01]	[0.00001, 0.001]	[0.0, 0.1]
# conv layers	{1, 2}	{2, 3}	{4, 5}	{1, ..., 5}
# filters	10-150	50-150, 150-300	50-150, 150-300, 300-400	10-150, 10-300, 10-500
	10-300	200-400	350-500, 400-600	10-600, 10-700
filter size	{3x3, 5x5, 7x7}	{3x3, 5x5, 7x7}	{3x3, 5x5}	{3x3, 5x5, 7x7}
# hidden layers	{1, 2, 3}	{1, 2}	{1, 2}	{1, 2, 3}
# hidden units	[100, 1000]	[300, 1000]	[500, 1000]	[100, 1000]
batch size	[10, 250]	[10, 130]	[10, 80]	[10, 150]

6 Best Hyperparameters per Algorithm (Experiment 2)

Here we provide a detailed overview of the hyperparameters that were found by the various optimization algorithms for the different data sets in the first experiment. Tables 18, 19, 20 and 21 describe the hyperparameters found for the 102 Flowers data set on the different resolutions and Table 17 describes the hyperparameters used during the evaluation on the test sets.

Table 17: **Test set errors** on the **102 Flowers** data set for input sizes of 128x128 pixels. The best results are highlighted in bold. Each setting was tested both without any additional regularization and with regularization methods such as dropout and batch normalization (bn).

Found by	Hyperparameters	Regularization	Val error

Genetic Algorithm	learning rate = 0.005, batch size = 20
	# conv layers = 5, filter size = [3,3,3,3,3]
	filters = [170,270,400,450,500]
	# hidden layers = 1, units = [900]
	L1 = 0.0001, L2 = 0.1
		none	0.536 ± 0.007

		dropout	0.532 ± 0.001
		dropout **bn**	**0.418 ± 0.005**

TPE	learning rate = 0.013, batch size = 20
	# conv layers = 5, filter size = [3,3,3,3,3]
	filters = [80,170,340,470,550]
	# hidden layers = 1, units = [1200]
	L1 = 0.0, L2 = 0.00001
		none	0.536 ± 0.005

		dropout	0.537 ± 0.006
		dropout **bn**	**0.415 ± 0.004**

SMAC	learning rate = 0.012, batch size = 27
	# conv layers = 5, filter size = [3,3,3,3,5]
	filters = [134,263,318,391,488]
	# hidden layers = 1, units = [727]
	L1 = 0.0, L2 = 0.0001
		none	0.538 ± 0.004

		dropout	0.545 ± 0.012
		dropout **bn**	**0.440 ± 0.003**

Random Search	learning rate = 0.012, batch size = 40
	# conv layers = 5, filter size = [3,3,3,5,3]
	filters = [60,250,540,570,550]
	# hidden layers = 1, units = [550]
	L1 = 0.0, L2 = 0.0
		none	0.546 ± 0.006

		dropout	0.551 ± 0.012
		dropout **bn**	**0.468 ± 0.006**

Table 18: Results of the best CNNs found by the **genetic algorithm** for the **102 Flowers** data set. The arrows between the input sizes for the genetic algorithm highlight that the search space of the genetic algorithm is initialized with the results of the genetic algorithm on the previous, smaller resolution.

Algorithm	Genetic Algorithm			
input size	32x32px →	64x64px →	128x128px	128x128px
# generations	15	10	5	30
avg gen time	28min 51sec	1h 32min 45sec	3h 51min 09sec	2h 14min 18sec
avg val error	0.692 ± 0.008	0.647 ± 0.004	0.640 ± 0.007	0.696 ± 0.008
learning rate	0.01 (100%)	0.01 (93%)	0.005 (60%), 0.01 (40%)	0.01 (93%
batch size	20 (27%) 30 (67%)	20 (47%) 30 (40%)	20 (67%) 30 (27%)	10-40 (27%) 50-70 (53%)
L1 reg	0.0 (100%)	0.0 (100%)	0.0 (100%)	0.0 (100%)
L2 reg	0.0 (47%) 0.001 – 0.01 (47%)	0.0 (60%) 0.0001 – 0.001 (40%)	0.0 (73%) 0.0001 – 0.001 (27%)	0.0 (47%) 0.0001 – 0.01 (53%)
# conv layers	3 (87%)	4 (80%), 5 (20%)	5 (93%)	4 (53%), 5 (40%)
# hidden layers	1 (100%)	1 (100%)	1 (100%)	1 (93%)
# filters				
1st conv layer	90 – 120 (60%) 160 – 180 (33%)	130 – 150 (100%)	160 – 180 (67%) 210 – 250 (33%)	30 – 50 (40%) 140 – 180 (40%)
2nd conv layer	100 – 120 (53%) 180 – 220 (47%)	150 – 210 (67%) 340 – 350 (33%)	270 – 280 (40%) 350 – 400 (33%)	200 – 300 (67%) 350 – 400 (27%)
3rd conv layer	130 – 150 (62%) 180 – 210 (23%)	320 – 370 (93%)	310 – 380 (33%) 410 – 440 (67%)	250 – 350 (47%) 400 – 450 (33%)
4th conv layer		350 – 390 (100%)	390 – 430 (40%) 460 – 490 (60%)	450 – 500 (40%) 550 – 600 (53%)
5th conv layer			440 – 470 (38%) 490 – 500 (54%)	550 – 600 (50%) 650 – 700 (50%)
filter size				
1st conv layer	3 (93%)	3 (100%)	3 (100%)	5 (33%), 7 (67%)
2nd conv layer	3 (100%)	3 (100%)	3 (100%)	5 (40%), 7 (40%)
3rd conv layer	3 (100%)	3 (100%)	3 (100%)	3 (60%), 5 (40%)
4th conv layer		3 (100%)	3 (100%)	3 (64%), 5 (36%)
5th conv layer		3 (100%)	3 (100%)	3 (83%)
# hidden units				
1st hidden layer	600 – 650 (60%) 900 – 950 (33%)	550 – 650 (53%) 850 – 950 (47%)	600 – 650 (33%) 800 – 950 (67%)	400 – 600 (27%) 900 – 1200 (47%)

Table 19: Results of the best CNNs found by **Random Search** for the **102 Flowers** data set. The arrows between the input sizes for the Random Search algorithm highlight that the search space of the Random Search algorithm is initialized with the results of the Random Search run on the previous, smaller resolution.

Algorithm	Random Search			
input size	32x32px →	64x64px →	128x128px	128x128px
# generations	15	10	5	30
avg gen time	12min 57sec	1h 44min 36sec	4h 42min 50sec	1h 39min 07sec
avg val error	0.728 ± 0.007	0.676 ± 0.009	0.673 ± 0.012	0.691 ± 0.007
learning rate	0.006 – 0.06 (100%)	0.01 – 0.08 (100%)	0.007 – 0.03 (100%)	0.001 – 0.03 (93%)
batch size	10 – 50 (27%) 60 – 120 (60%)	20 – 50 (40%) 80 – 110 (53%)	20 – 50 (53%) 60 – 80 (47%)	10 – 50 (47%) 60 – 100 (33%)
L1 reg	0.0 (100%)	0.0 (100%)	0.0 (100%)	0.0 (100%)
L2 reg	0.0 (47%) 0.001 – 0.01 (33%)	0.0 (53%) 0.0001 – 0.0005 (40%)	0.0 (40%) 0.0001 – 0.009 (47%)	0.0 (47%) 0.001 – 0.03 (27%)
# conv layers	2 (100%)	3 (40%), 4 (60%)	4 (53%), 5 (47%)	4 (33%), 5 (60%)
# hidden layers	1 (93%)	1 (93%)	1 (87%)	1 (93%)
# filters				
1st conv layer	40 – 80 (33%) 100 – 150 (67%)	50 – 100 (47%) 110 – 150 (53%)	60 – 120 (53%) 130 – 170 (47%)	60 – 100 (33%) 110 – 150 (67%)
2nd conv layer	100 – 150 (33%) 200 – 300 (47%)	150 – 250 (60%) 300 – 350 (27%)	150 – 250 (80%) 280 – 350 (20%)	120 – 170 (47%) 180 – 230 (40%)
3rd conv layer		450 – 550 (80%)	300 – 400 (40%) 400 – 550 (60%)	240 – 300 (47%) 310 – 350 (47%)
4th conv layer		300 – 500 (56%) 550 – 650 (33%)	500 – 600 (60%) 610 – 700 (33%)	350 – 400 (57%) 410 – 480 (36%)
5th conv layer			500 – 600 (43%) 700 – 800 (57%)	350 – 450 (33%) 460 – 550 (67%)
filter size				
1st conv layer	3 (73%), 5 (13%)	3 (93%)	3 (47%), 5 (33%)	3 (60%), 5 (27%)
2nd conv layer	3 (53%), 5 (33%)	3 (93%)	3 (67%), 5 (33%)	3 (47%), 5 (40%)
3rd conv layer		3 (27%), 7 (53%)	3 (53%), 5 (47%)	3 (53%), 7 (40%)
4th conv layer		3 (67%)	5 (47%), 7 (33%)	3 (33%), 5 (40%)
5th conv layer			3 (57%), 7 (29%)	3 (44%), 7 (33%)
# hidden units				
1st hidden layer	300 – 500 (53%) 700 – 900 (40%)	400 – 600 (73%) 700 – 800 (27%)	450 – 650 (60%) 700 – 800 (40%)	450 – 650 (33%) 700 – 900 (47%)

Table 20: Results of the best CNNs found by **TPE** for the **102 Flowers** data set. The arrows between the input sizes for the TPE algorithm highlight that the search space of the TPE algorithm is initialized with the results of the TPE run on the previous, smaller resolution.

Algorithm	TPE			
input size	32x32px →	64x64px →	128x128px	128x128px
# generations	15	10	5	30
avg gen time	37min 06sec	1h 22min 40sec	3h 47min 17sec	2h 07min 24sec
avg val error	0.698 ± 0.006	0.652 ± 0.012	0.657 ± 0.013	0.654 ± 0.009
learning rate	0.005 – 0.05 (100%)	0.002 – 0.01 (100%)	0.005 (33%), 0.01 (67%)	0.001 – 0.005 (100%)
batch size	10 – 30 (73%) 60 – 70 (20%)	10 – 20 (80%) 30 – 50 (20%)	30 (53%) 40 (47%)	10 (73%) 20 (20%)
L1 reg	0.0 (100%)	0.0 (100%)	0.0 (100%)	0.0 (100%)
L2 reg	0.0 (20%) 0.001 – 0.01 (67%)	0.0 (27%) 0.0001 – 0.001 (67%)	0.0 (67%) 0.0001 – 0.005 (27%)	0.0 (60%) 0.0005 – 0.01 (33%)
# conv layers	2 (100%)	3 (33%), 4 (67%)	4 (67%), 5 (33%)	4 (33%), 5 (67%)
# hidden layers	1 (100%)	1 (100%)	1 (93%)	1 (100%)
# filters				
1st conv layer	40 – 70 (53%) 90 – 140 (47%)	80 – 100 (60%) 120 – 140 (40%)	80 – 110 (60%) 120 – 140 (40%)	20 – 50 (33%) 120 – 150 (67%)
2nd conv layer	180 – 220 (33%) 240 – 280 (67%)	180 – 210 (60%) 220 – 240 (33%)	150 – 200 (27%) 210 – 250 (73%)	50 – 70 (33%) 130 – 170 (67%)
3rd conv layer		260 – 300 (60%) 330 – 360 (40%)	260 – 300 (47%) 320 – 350 (53%)	200 – 300 (33%) 350 – 450 (53%)
4th conv layer		430 – 470 (100%)	430 – 450 (47%) 480 – 550 (53%)	350 – 400 (47%) 540 – 570 (40%)
5th conv layer			520 – 550 (100%)	40 – 70 (50%) 450 – 500 (50%)
filter size				
1st conv layer	3 (100%)	3 (100%)	3 (100%)	3 (100%)
2nd conv layer	3 (73%), 5 (27%)	3 (100%)	3 (100%)	3 (100%)
3rd conv layer		3 (100%), 7 (33%)	3 (67%)	3 (100%)
4th conv layer		3 (100%)	3 (67%), 5 (33%)	3 (47%), 7 (47%)
5th conv layer			3 (100%)	3 (50%), 5 (50%)
# hidden units				
1st hidden layer	400 – 600 (47%) 800 – 1000 (53%)	600 – 800 (33%) 900 – 1200 (67%)	800 – 1000 (60%) 1100 – 1300 (40%)	400 – 600 (33%) 750 – 900 (67%)

Table 21: Results of the best CNNs found by **SMAC** for the **102 Flowers** data set. The arrows between the input sizes for the SMAC algorithm highlight that the search space of the SMAC algorithm is initialized with the results of the SMAC run on the previous, smaller resolution.

Algorithm	SMAC			
input size	32x32px	64x64px	128x128px	128x128px
# generations	15	10	5	30
avg gen time	14min 07sec	1h 10min 12sec	3h 51min 11sec	1h 11min 15sec
avg val error	0.729 ± 0.005	0.676 ± 0.006	0.653 ± 0.006	0.659 ± 0.008
learning rate	0.005 – 0.05 (100%)	0.003 – 0.03 (100%)	0.005 – 0.02 (100%)	0.001 – 0.006 (93%)
batch size	10 – 30 (53%) 80 – 100 (47%)	10 – 20 (47%) 40 – 80 (47%)	20 – 30 (67%) 40 – 60 (27%)	10 – 20 (60%) 30 – 40 (40%)
L1 reg	0.0 (100%)	0.0 (100%)	0.0 (100%)	0.0 (100%)
L2 reg	0.0 (67%) 0.0009 – 0.0002 (33%)	0.0 (67%) 0.003 – 0.008 (33%)	0.0 (100%)	0.0 (67%) 0.001 – 0.01 (33%)
# conv layers	2 (100%)	3 (100%)	5 (93%)	4 (87%), 5 (13%)
# hidden layers	1 (93%)	1 (100%)	1 (100%)	1 (100%)
# filters				
1st conv layer	80 – 90 (60%) 100 – 110 (40%)	50 – 80 (40%) 100 – 120 (53%)	80 – 100 (40%) 120 – 150 (47%)	30 – 80 (67%) 130 – 140 (33%)
2nd conv layer	160 – 200 (67%) 210 – 250 (27%)	150 – 180 (73%) 250 – 300 (27%)	180 – 220 (60%) 250 – 280 (33%)	10 – 20 (33%) 200 – 300 (53%)
3rd conv layer		350 – 400 (87%)	320 – 360 (93%)	180 – 220 (47%) 420 – 450 (40%)
4th conv layer			380 – 420 (67%) 440 – 480 (33%)	350 – 440 (47%) 550 – 600 (47%)
5th conv layer			470 – 520 (60%) 580 – 600 (27%)	
filter size				
1st conv layer	9 (80%)	3 (87%)	3 (87%)	3 (93%)
2nd conv layer	5 (33%), 7 (47%)	3 (93%)	3 (87%)	3 (87%)
3rd conv layer		3 (67%), 5 (33%)	3 (100%)	3 (100%)
4th conv layer			3 (100%)	5 (47%), 7 (47%)
5th conv layer			3 (87%)	
# hidden units				
1st hidden layer	300 – 500 (47%) 800 – 1000 (53%)	500 – 750 (33%) 800 – 1000 (60%)	650 – 750 (80%) 800 – 900 (20%)	300 – 500 (27%) 600 – 800 (73%)

References

[1] Hossein Azizpour, Ali Sharif Razavian, Josephine Sullivan, Atsuto Maki, and Stefan Carlsson. From Generic to Specific Deep Representations for Visual Recognition. In *Conference on Computer Vision and Pattern Recognition Workshops*, pages 36–45, 2015.

[2] Yoshua Bengio. Practical Recommendations for Gradient-Based Training of Deep Architectures. In *Neural Networks: Tricks of the Trade*, pages 437–478, 2012.

[3] James Bergstra, Rmi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems 24*, pages 2546–2554, 2011.

[4] James Bergstra, Dan Yamins, and David Cox. Hyperopt: A Python Library for Optimizing the Hyperparameters of Machine Learning Algorithms. In *Python in Science Conference*, pages 13–20, 2013.

[5] James Bergstra, Daniel Yamins, and David Cox. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. In *International Conference on Machine Learning*, pages 115–123, 2013.

[6] Thomas M. Breuel. The Effects of Hyperparameters on SGD Training of Neural Networks. *arXiv:1508.02788*, 2015.

[7] Adam Coates, Andrew Y Ng, and Honglak Lee. An Analysis of Single-Layer Networks in Unsupervised Feature Learning. In *International Conference on Artificial Intelligence and Statistics*, pages 215–223, 2011.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[9] Pooya Khorrami, Thomas Paine, and Thomas Huang. Do Deep Neural Networks Learn Facial Action Units When Doing Expression Recognition? In *IEEE International Conference on Computer Vision*, pages 19–27, 2015.

[10] Patrick Lucey, Jeffrey Cohn, Takeo Kanade, Jason Saragih, Zara Ambadar, and Iain Matthews. The Extended Cohn-Kanade Dataset (CK+): A complete dataset for action unit and emotion-specified expression. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 94–101, 2010.

[11] Maria-Elena Nilsback and Andrew Zisserman. Automated Flower Classification over a Large Number of Classes. In *Conference on Computer Vision, Graphics Image Processing*, pages 722–729, 2008.

[12] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. In *International Conference on Learning Representations*, 2014.

[13] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. CNN Features Off-the-Shelf: An Astounding Baseline for Recognition. In *Conference on Computer Vision and Pattern Recognition*, pages 806–813, 2014.