Computational Intelligence ( )

Submission Assignment 5

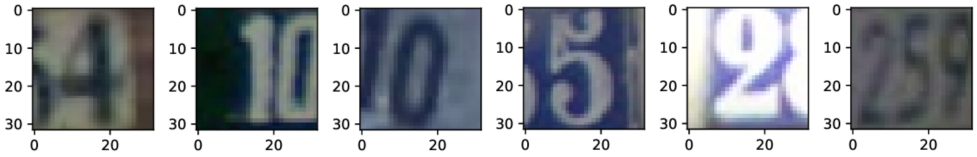
Instructor: Jakub M. Tomczak *Name:* Caspar M. S. Grevelhoerster, *Netid:* 2707848

1 Introduction

Evolutionary algorithms (EA) apply nature-inspired mechanisms of evolution to problems utilizing the advantageous behaviors of life. EAs apply a sequence of steps, namely selection, recombination and mutation that together aid an incremental convergence from randomly initiated genotypes representing arbitrary design choices to highly customized genotypes which habituate to the problem space over time. The main advantage of these algorithms is the multitude of possible fields of research in which they are applicable.

The ubiquitous problem of image classification is commonly solved with convolutional neural networks (CNN). Due to their complex structure allowing for a variety of different architectures, they must be carefully and thoroughly tailored to the problem at hand. Even though this labor-intensive task is commonly done by human designers that tirelessly optimize CNN architectures, automating this endeavor through neuroevolution has been a popular topic of research. EAs have been commonly applied in the design neural network architectures due to their aforementioned high level of adaptability. Decades ago, long before CNNs had been invented, Miller et al. (1989) among others researched the effectiveness of EAs in such applications and found that they are indeed very effective when designing neural networks.

Figure 1: Examples of picture data contained in the SVHN dataset



In this paper, an EA algorithm will be described that is designed to adjust the architecture of a CNN network such that the neural network can be used to classify 32 by 32 pixel RGB-image files (fig. 1) of house numbers (Netzer et al., 2011). An advanced CNN will present a comparison for resulting classification performance. Further, the specifically customized evolutionary procedures that together make up the EA algorithm will be unfolded and possible improvements discussed.

2 Problem statement

Neural networks (NN) facilitate the recognition of patterns in data by learning complex correlations. This learning happens based on a penalty (loss function) whose value the network minimizes over time by updating the way it pays attention (weight updates through backpropagation). In the case of the SVHN dataset, the applied CNNs classify pictures into the ten distinct digits by means of assigning class probability values to their output nodes. These class probabilities in shape of a vector (e.g. $[0.1, 0.3, 0.6]$ in the case of three classes) can be compared to the actual (target) class represented by a vector with one-hot encoding of the class (e.g. $[0, 0, 1]$). The negative log likelihood loss $\text{NLLL}(\Theta) = -\sum_{n=1}^{N}(y_n \ln s_n + (1 - y_n)\ln(1 - s_n))$ is applied as a loss function for the CNNs in this paper. It computes the distance between the probabilities assigned to the class labels and the target vector representing the true value. Not only does it thereby punish wrong classifications, it also punishes low confidence levels:

$$\text{NLLL}(\begin{matrix} y = [0,0,1], \\ s = [0.1,0.1,0.8] \end{matrix}) \approx 0.1 < \text{NLLL}(\begin{matrix} y = [0,0,1], \\ s = [0.1,0.3,0.6] \end{matrix}]) \approx 0.22$$

This loss is minimized over time such that the difference between predicted and actual label become as small as possible. After some number of iterations, a CNN can be applied to classify unseen pictures since it has learned to produce predictive class probabilities that lie close to the truth. Since the objective of the EA, however, does

not depend on the networks' classification confidence but rather their accuracy, only the training is based on NLLL, not the evaluation-of-fitness function. The fitness is calculated based on the classification error produced by the network, so the simple sum of the distances between one-hot predictions and one-hot targets of one batch, disregarding confidence levels.

The extent to which a CNN can produce accurate class predictions is highly dependent on its structural components as well as their arrangement, i.e. architecture. The process of choosing out of countless possible components and specifying complex hyper parameters that make up an optimal CNN architecture is time intensive and complex. Applying an EA to explore the broad search space of components and their parameters is therefore an obvious choice. As mentioned in section 1, the EA is an artificial simulation of evolution which selects individuals based on their fitness in an environment. In this particular problem, the classification problem resembles the environment in which CNN-architecture-individuals are selected, recombined and mutated based on their ability to classify. Since this evolutionary process prefers individuals with a better (lower) fitness, those architectures that have a better classification ability will dominate the population. Over time, this behavior leads to the EA generating increasingly good network structures. The objective function f of the EA is therefore the fitness of CNN-architecture-individuals, i.e. their classification error (CE) produced while classifying images of the validation dataset. In order to keep training times low, the task specified to penalize the number of trainable parameters (weights) of each CNN, thereby impeding particularly large networks in their ability to compete.

$$f(g) = ce(g) + \lambda * \frac{N_p}{N_{\max}}, \text{ where } ce(g) = ce \text{ from training CNN given } g$$

With this objective function in place, the classification errors ce of the competing CNNs are increased (which is worse given the minimization objective) if their number of trainable parameters N_p is high relative to the maximal number of parameters of the largest network $N_{\max}$. The EA algorithm will find a CNN structure that is well adjusted to the objective function $f(g)$ by classifying with low error while maintaining a minimal number of weights.

3 Methodology

In order to limit the search space, the criteria for this paper specify that the CNN architecture must be made up of the following sequence of layers:

Pixel Input ⇒ Convolutional Layer → Activation Function → Pooling Layer → Flatten Layer → Linear Layer → Activation Function → Linear Layer → LogSoftMax Layer ⇒ *Class Prediction*

Within this constrained architecture, there are a number of hyper-parameters that represent the pool of choices available to the EA. In order to maximize the CNNs' classification performances, the set of hyper-parameters, represented by a vector of integers, needs to be optimized.

A Convolutional Layer contains a set of kernels of which each produces an activation mapping (also called filters, channels) from the current set of input channels (i.e. red, green, blue for the described dataset). It is defined that this layer may have a number of output filters of either 8, 16, or 32. Further, it can have kernels of size $(5, 5)$ with padding set to 2 or kernel size $(3, 3)$ with padding 1. Stride is kept at 1 in any case.

An Activation Function enables a neural network to learn representations beyond linear complexity (without them, NN would be complex representations of simple linear regression). There are two positions at which activation functions are used and since the task does not clearly specify if both should be of the same type, it was decided to make them individual types, enabling two different layer types.

A Pooling Layer compares a number of adjacent node values (also: kernel size or window size) of a filter and copies the maximum or average value to a new filter, thereby reducing noise and the number of features. One choice is between pooling the average or the maximum value and the second between a window width and height of 1 or 2.

After the Flatten Layer, that enqueues all output features of the previous layer (that are stored in multi-dimensional tensors) to a one-dimensional vector, two Linear Layers are set in place. These fully connect all output features from the previous layer to a specified number of output features by means of weights. The choice lies in the number of nodes that will represent the input for the second linear layer that follows the first. That number of nodes $n \in \mathbb{N}, n \in (10, 100), 10|n$.

The LogSoftMax Layer is set in place in order to normalize the values of the output nodes of the last linear layer to class probabilities $\in (0, 1)$ whose sum $= 1$. The logarithmic softmax-activation applies to x through

$$\text{LogSoftMax}(x_i) = \log(\frac{\exp(x_i)}{\sum_j \exp(x_j)}).$$

All of these hyper-parameters are represented in vector format and thereby uniquely identify the architecture of an individual CNN.

$$\forall g, g \in \begin{bmatrix} \text{Nr. of conv. filters } g_0, & g_0 \in \mathbb{N}, 0 \leq g_0 \leq 3 \\ \text{K. size \& padding } g_1, & g_1 \in \mathbb{N}, 0 \leq g_1 \leq 1 \\ \text{Activ. func. 1 } g_2, & g_2 \in \mathbb{N}, 0 \leq g_2 \leq 4 \\ \text{Max/avg p., w. size } g_3, & g_3 \in \mathbb{N}, 0 \leq g_3 \leq 3 \\ \text{Nr. nodes lin. } g_4, & g_4 \in \mathbb{N}, 0 \leq g_4 \leq 9 \\ \text{Activ. func. 2 } g_5, & g_5 \in \mathbb{N}, 0 \leq g_5 \leq 4 \end{bmatrix}$$

The above vector can be seen as the genotype g of the individuals, encoding their phenotype (in this case classification performance) in integer format. There are a total number of $4 * 2 * 5 * 4 * 10 * 5 = 8000$ possible combinations that make up the search space explored by the EA through four main methods that facilitate exploration process:

1. Explorative Population Initialization (Rahnamayan et al., 2007). From n (where n is equal to the population size) randomly generated genotypes g (within the bounds described above), a vector b containing the upper bounds $[3, 1, 4, 3, 9, 4]$ is subtracted, resulting in vector $\hat{g}$ that represents an individual positioned exactly opposite of g. Subsequently, all genotypes that are part of the union of both sets $\{g_1, g_2, ..., g_n\} \cup \{\hat{g}_1, \hat{g}_2, ..., \hat{g}_n\}$ are evaluated based on their fitness using the objective function $f(g)$ and n individuals are selected, forming generation 0. Before even applying any evolutionary step, this method helps exploring twice the number of positions in the search space, thereby increasing the convergence speed of the EA.

2. Recombination. In order to recombine genes, parental genotypes are sorted by performance. The ith and i+1th (where step size of i is 2 and bounds are 0 and n) parents are selected and at a random number of random pointer positions, the genotypes are crossed over. This produces two offspring genotypes that contain a mutually exclusive set of random combination of genes, maintaining order. Due to the iteration over parents which are ranked by performance, genes of those with similar performance are recombined, which is advantageous.

3. Mutation. Each offspring genotype produced by the recombination method is randomly mutated based on a certain probability. The mutation of a genotype vector works by altering a randomly chosen position (gene) to a new random integer within the respective bounds described above. The mutation process facilitates exploration of the search space as well preventing premature convergence to a local minimum of the population positions by possibly choosing distant gene values by chance.

4. Survivor Selection. Each individual that is part of the union of the offspring (including those individuals whose genes might have been altered through mutation) and parents is evaluated based on the objective function $f(g)$ (phenotype). All individuals are ranked based on their phenotype and the better half is passed on to the next generation. This method is called elitism since it compares performance of both parents and children and simply takes the better performing half of individuals.

Over time, the algorithm will close in on the optimal CNN architecture. After finishing the computation of a number of generations, the best genotype vector g can be returned, properly trained for a greater number of iterations (since now, computational expense does not affect the EA's runtime) and finally its classification performance tested on the testing data that is thus far set aside for this purpose.

4 Experiments

There are several experimental EA-hyper-parameters or properties (not regarding the CNN layer structure) that were specified during the experimental setup. Because of the relatively large input size of 32 by 32 pixel images, large CNNs genotypes can encode architectures of maximal size of 3280342 total trainable parameters (weights), which is represented by the parameter vector $[2, 1, 0, 1, 9, 0]$. With this high number of weights, training times are extensive, especially since the EA evaluates each parent and each child once per iteration. Hence, the population size n is set to 10, which levels out the training time and the extent to which the EA can explore the search space, which correlates with n.

The second hyper-parameter was the number of epochs that each CNN was trained for. Setting this value to a low value of 3 selects CNNs that can quickly converge to a low fitness also marginally penalizes large CNN

networks (they need longer to converge because there is more data to comprehend). Three iterations are also enough to make out drastic differences in classification performance of the different architectures.

The last two hyper-parameters are both set with the same incentive of increasing the chance of mutation per child. The mutation probability was set to 0.6, which strikes a balance between mutating too many of the recombined parental genotypes (this would make recombination superfluous) and maximizing exploration through random alteration (thereby possibly enabling population to escape local minima). Setting the number of generations to 50 also allows for more opportunities of mutation, possibly leading to a quicker converging CNN by chance.

In order to test the performance of the EA, an experiment will be explained that shows the convergence of the population towards good performance, thereby confirming the choices in hyper-parameters described above. Further, the EA-optimized CNN architecture will be trained on a number of epochs and finally evaluated on the testing data. These results can then be compared to a highly optimized CNN network exhibiting optimal training performance that was designed during earlier stages of this project. Thereby, limitations of both the provided CNN as well as the EA can be discussed.

5 Results and discussion

The six matrices depicted below show all genes combined to a genotype in one row and their fitness values in the respective entries of the right most column. The population was extracted from the EA during run-time and aim to show the convergence towards features that encode good fitness.

0:
$$\left[\begin{array}{cccccc|c} 1&1&2&3&7&4&0.18\\ 2&1&4&1&8&3&0.18\\ 1&1&4&1&6&3&0.19\\ 1&1&2&3&8&3&0.20\\ 2&1&4&1&7&0&0.21\\ 1&0&0&0&9&1&0.21\\ 1&0&0&0&9&0&0.22\\ 1&0&2&3&9&0&0.24\\ 1&1&4&1&6&1&0.30\\ 1&0&2&3&9&1&0.33 \end{array}\right]$$

10:
$$\left[\begin{array}{cccccc|c} 2&1&2&2&8&3&0.17\\ 1&1&2&2&8&3&0.17\\ 1&1&2&2&8&3&0.17\\ 1&1&2&2&8&3&0.17\\ 1&1&4&1&8&3&0.18\\ 1&1&2&3&7&4&0.18\\ 1&1&2&2&8&3&0.18\\ 2&1&4&1&8&3&0.18\\ 1&1&4&2&8&0&0.19\\ 0&1&4&1&7&4&0.19 \end{array}\right]$$

20:
$$\left[\begin{array}{cccccc|c} 2&1&2&2&8&3&0.15\\ 2&1&2&2&8&3&0.16\\ 2&1&2&2&8&3&0.16\\ 2&1&2&2&8&4&0.16\\ 2&1&2&2&8&3&0.16\\ 2&1&2&2&8&3&0.16\\ 2&1&2&2&6&3&0.16\\ 2&1&2&2&8&3&0.16\\ 2&1&2&2&8&3&0.16\\ 2&1&2&2&8&3&0.16 \end{array}\right]$$

30:
$$\left[\begin{array}{cccccc|c} 2&1&2&2&9&3&0.15\\ 2&1&2&2&8&4&0.15\\ 2&1&2&2&8&3&0.15\\ 2&1&2&2&8&4&0.15\\ 2&1&2&2&8&3&0.15\\ 2&1&2&2&8&4&0.15\\ 2&1&2&2&8&3&0.16\\ 2&1&2&2&8&3&0.16\\ 2&1&2&2&8&3&0.16\\ 2&1&2&2&8&3&0.16 \end{array}\right]$$

40:
$$\left[\begin{array}{cccccc|c} 2&1&2&2&9&3&0.15\\ 2&1&2&2&8&4&0.15\\ 2&1&2&2&9&4&0.15\\ 2&1&2&2&8&3&0.15\\ 2&1&2&2&8&3&0.15\\ 2&1&2&2&8&3&0.15\\ 2&1&2&2&8&4&0.15\\ 2&1&2&2&8&3&0.15\\ 2&1&2&2&8&4&0.15\\ 2&1&2&2&8&4&0.15 \end{array}\right]$$

50:
$$\left[\begin{array}{cccccc|c} 2&1&4&2&9&4&0.15\\ 2&1&2&2&8&4&0.15\\ 2&1&2&2&8&4&0.15\\ 2&1&2&2&9&3&0.15\\ 2&1&2&2&8&4&0.15\\ 2&1&2&2&8&4&0.15\\ 2&1&2&2&9&4&0.15\\ 2&1&2&2&8&3&0.15\\ 2&1&2&2&8&3&0.15\\ 2&1&2&2&8&3&0.15 \end{array}\right]$$

As becomes evident from the genotypes above, the EA algorithm succeeds in finding an advantageous combination of features. At generation 0, the explorative method for the population initialization leads to a selection of genotypes that already have decent fitness values. In comparison to a fully random method producing fitnesses with a median average of between 30 and 40 over multiple initializations, an average fitness of 0.23 facilitates the exploration process through an initial performance boost.

Over time, the different individuals tend to have consistent values for the same genotype. This behavior is best seen in generation 30 and 40, where only small deviations exist. Through mutation, an even better individual with a fitness value of about 0.145 is found in the last generation of the EA (genotype [2,1,4,2,9,4]) that differs from the rest of the population in the type of activation function used at the first of the two available positions. Genotype [2,1,4,2,9,4] encodes the following final CNN structure:

→ Convolutional Layer with 32 filters, square kernel size 2 and padding 2.
→ ELU Activation Function 1
→ Max Pooling with kernel size 2
→ Flatten

→ Linear layer with 8192 input and 100 output features
→ ELU Activation Function 2
→ Linear layer with 100 input and 10 output features
→ LogSoftMax

Resetting this structure to random feature values and training it on the training dataset yielded a final classification error rate of 0.152 on the test set. The plots below depict the performance on the validation data during training as well as the final testing performance computed on the testing data by the dashed line and its label above the line on the right.

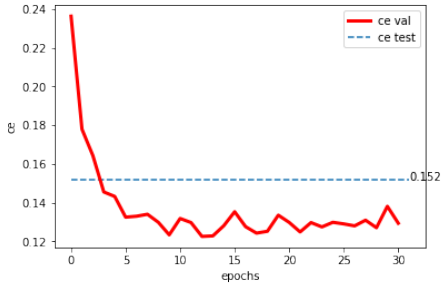


Figure 2: Final CNN. Classification error.

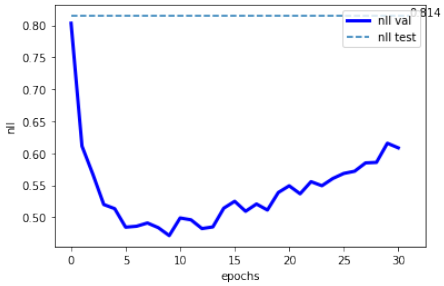


Figure 3: Final CNN. Negative Log Likelihood Loss.

The plots show that the model reaches its best performance (so low values) after around 9 epochs. Afterwards, performance on the validation data decreases, which is indicated by increasing values in both plots after the 9th epoch. The fact that the performance indicated by the dashed line in both plots is so much worse than the performance during training as well as the mentioned rising NLLL value both indicate that the CNN is overfitting on the testing data. Later testing shows that adding dropout layers to the CNN structure helps settling this issue, however, since this was not part of the provided architecture, this improvement should instead be subject of future research.

To test whether or not only the [2,1,4,2,9,4] genotype exhibits overfitting behavior, results of another experiment are depicted below.

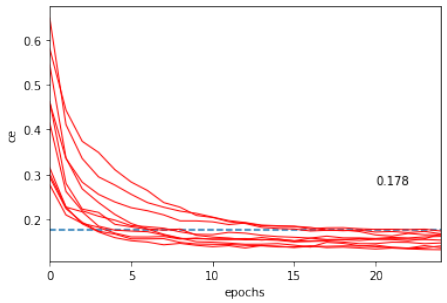


Figure 4: Population 0. Classification error.

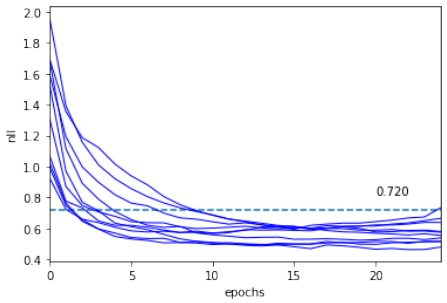


Figure 5: Population 0. Negative Log Likelihood Loss.

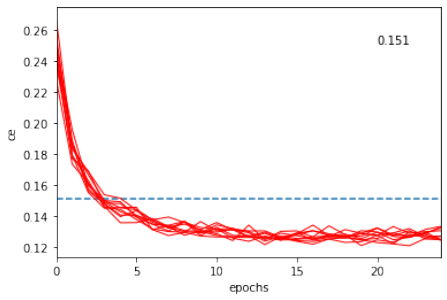


Figure 6: Population 50. Classification error.

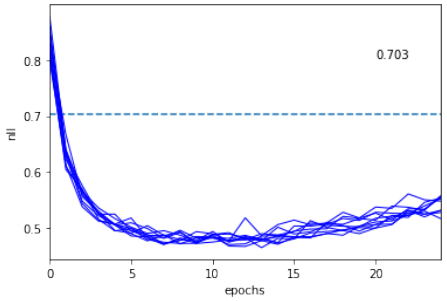


Figure 7: Population 50. Negative Log Likelihood Loss.

As figure 4 and 5 show, the training performance of all CNNs of population 0 show healthy convergence behavior. Both mean testing error as well as mean testing NLLL are not significantly higher than values

during training. Additionally, only one individual seems to overfit by slightly worsening its performance during later epochs. Figure 6 and 7 of generation 50 on the other hand, are evidence for the EA putting selection pressure on high, immediate performance and eventual overfitting behavior. However, the difference in y-scale of the plots are still evidence for high performance that is being selected for in the CNN architectures. A possible explanation for the behavior of incremental loss values and a stable error rate during training might be that wrong predictions have increasingly lower confidence for the true class label. This could be produced by the characteristic of NLLL penalizing bad predictions more strongly than good predictions are rewarded. The true reason for this behavior, however, would need to be explored in future research.

Lastly, as was mentioned in section 1, the EA-optimized CNN (eaCNN) architecture will be compared to the architecture of a more advanced CNN:
Pixel Input ⇒
Convolutional Layer, Kernel size 3, 32 Filters → ReLU Activation → MaxPooling Layer, Window size 2 → Dropout, Probability 0.3 → Convolutional Layer, Kernel size 3, 64 Filters → ReLU Activation → MaxPooling Layer, Window size 2 → Dropout, Probability 0.3 → Convolutional Layer, Kernel size 3, 128 Filters → ReLU Activation → MaxPool, Window size 2 → Dropout, Probability 0.3 → Flatten → ReLU Activation → Dropout, Probability 0.4 → Linear Layer, 2048 Inputs, 10 Outputs → LogSoftMax Layer
⇒ *Class Prediction*
This highly optimized CNN (hoCNN) shows the following training CE and NLLL.

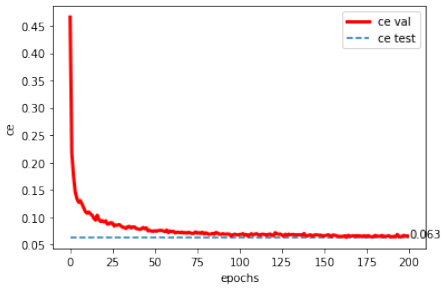


Figure 8: hoCNN. Classification error.

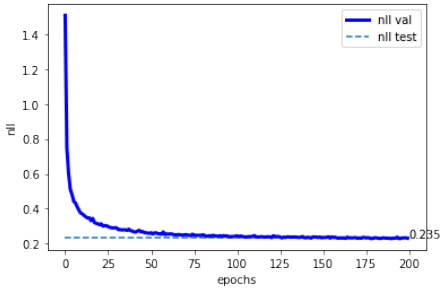


Figure 9: hoCNN. Negative Log Likelihood Loss.

Both the CE in figure 8 as well as NLLL in figure 9 produced on the testing data (dashed blue lines) are not above the validation curves but rather the margin to which the network performance converges. The convergence curves show a healthy correlative behavior and they are of monotonic decreasing negative exponential shape which is ideal.

Comparing the eaCNN to this performance might seem unjustifiable given the larger number of layers and the inclusion of Dropout layers in the hoCNN structure. However, there are a number of conclusions that can be drawn given the drastic difference in performance.

If an EA is based on a computationally expensive objective function, computation time will be very high since every new individual needs to be compared to the older ones from previous generations. In the case of NN architecture optimization, the only real way to test performance of a genotype is to train it and validate it on validation data. Each hyper-parameter that is added by analyzing more number of layers and more layer types, broadens the search space significantly. This is, because all already existing possible combinations of genes are multiplied by the different values a hyper-parameter can take (i.e. the number of options it encodes in the architecture). A broader search space self-evidently requires a higher population size of the EA and more layers (leading to more trainable parameters) extend the evaluation time of the NNs. In addition, it is important to point out that EAs work in a stochastical manner. This means that no design can guarantee convergence towards the best genotype. It is highly complex to design good hyper-parameters that ensure consistent convergence and even with the same circumstances, EAs might converge to different minima in the search space. Lastly, it could be interesting to explore how big the impact is of the weight penalty on the objective function given that the number of training epochs for each CNN is set to only 3.

For the future, it might additionally be interesting to design a neural network for classification that predicts the performance of another network given its genotype. Training such a performance predictor will take a long time, but once set up and saved, such model would make the evaluation of the objective function virtually instant given that the necessity to train a new network with a new genotype is eliminated. This approach would be interesting to explore in future research.

References

Miller, G. F., Todd, P. M., and Hegde, S. U. (1989). Designing neural networks using genetic algorithms. In *ICGA*, volume 89, pages 379–384.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning.

Rahnamayan, S., Tizhoosh, H. R., and Salama, M. M. (2007). A novel population initialization method for accelerating evolutionary algorithms. *Computers & Mathematics with Applications*, 53(10):1605–1614.