Symposium MATHEMATICS AND APPLICATION, Faculty of Mathematics, University of Belgrade, 2019 ,Vol. X (1)

Modified Genetic Algorithm for Learning Deep Convolutional Architectures

Milan M. Čugurović

Faculty of Mathematics, University of Belgrade, Studentski trg 16, 11000 Belgrade
e-mail: milan_cugurovic@matf.bg.ac.rs

Nikola Dimitrijević

Microsoft Development Center Serbia, Španskih boraca 3, 11000 Belgrade
e-mail: nikoladim95@gmail.com

Stefan Mišković

Faculty of Mathematics, University of Belgrade, Studentski trg 16, 11000 Belgrade
e-mail: stefan@matf.bg.ac.rs

Abstract.

This paper presents a modified variant of genetic algorithm for learning convolutional architectures and their training, which reduces the execution time of the algorithm. Modification is based on changing the evolutional segment of the algorithm by focusing on limiting the training time of each individual and incorporating the obtained knowledge of neuron parameters from the previous generations into each new one. By doing so the evolution is made more efficient, thus reducing the time needed to find the desired architecture.

Additional contribution of this paper is creating new dataset *DoubledMNIST*, which represents a successor of the popular MNIST dataset. Created dataset is doubled with respect to the MNIST dataset both in terms of the number of instances and in terms of the resolution of each individual instance. Results shown in the paper were obtained using the presented improved method on the created dataset. In this paper, classification results on the mentioned dataset are also presented.

Keywords: Genetic algorithm; CNN architectures; MNIST dataset; DoubledMNIST dataset

1. Introduction

Inspired by modern trends in development of deep convolutional neural architectures [28, 29], researches are attempting to find systems that will automatically generate suitable architectures for deep neural networks. Those architectures should on one hand have enough capacity (in terms of learning ability respectively number of parameters), while on the other hand, they should be easy for training and gradient propagation. One way to approach solving this problem is to leverage genetic algorithms which simulate evolution that favors architectures that are able to adapt fast and fulfill former requirements. This approach yields very promising results [5, 30, 31]. For some specific areas good results are achieved by combining convolutional neural networks with genetic algorithms [2, 32], which shows that they are highly compatible.

On the other hand, evolutionary algorithms, which are based on simulating natural selection, and therefore require large number of generations and additionally large number of individuals in each generation are, by their very nature, complementary with deep learning architectures. Concretely, with deep convolutional neural networks, given that they have many parameters, which is why they require great amount of time to be optimized. In this paper, we showed that there are ways to mitigate these compatibility gaps in the techniques mentioned.

We also created a new dataset for this purpose, which we believe will become the successor to the now-viral MNIST dataset. Our created dataset is duplicated with respect to MNIST in terms of the number of images as well as the dimensions of each, so modifying architectures that work well on MNIST to work well on our dataset is a natural step of pedagogical introduction into the field of machine learning. In addition, the dimensions of the created dataset make it competitive for realistic, industrial applications in the field of handwriting recognition.

Chapter 2 gives an overview of related work in the field, Chapter 3 details the dataset we created, Chapter 4 explains genetic representation of a neural network, Chapter 5 discusses the proposed improvement method, while Chapter 6 gives specific results obtained by evaluation of the proposed approach. Chapter 7 concludes, recapitulates what has been done and presents further guidance for future improvements.

2. Related work

This section provides background regarding offline handwritten character datasets and incorporating genetic algorithms with the learning of CNN architectures, and their training.

2.1. Offline handwriting datasets

The existence of quality datasets of handwritten characters is a prerequisite for the development of quality handwriting recognition techniques and their evaluations in different research scenarios. The task of collecting this kind of data is very demanding and laborious, since in order to create a dataset that includes a variety of writing styles, a large number of respondents must be involved. The process of creating handwritten databases began in the 1990s [9] and is still ongoing. In the meantime, a large number of datasets have been developed.

Some of the most important and commonly used datasets of offline handwritten characters are: CVL database [18], RIMES database [10], IAM dataset [11], NIST [12, 13], MNIST [8] and EMNIST [1] datasets, CEDAR [14], UNIPEN [15], IBM UB [16] and so on. Although the most commonly used datasets are datasets of handwritten Latin/English alphabet, there are significant datasets from other alphabets too. Here we refer to datasets of handwritten Chinese [19, 21, 20], Arabic [22, 23], as well as Bangala [24, 25] datasets, while there are many others. In addition, the development of multilingual datasets has been intensively developing in the last few years [17, 26, 18], which seeks to create an alphabet-agnostic handwriting recognition system. A more detailed overview of handwritten character datasets can be found in [9], while the datasets of the NIST family will be described in more detail below, since the dataset we created was built on the same basis.

2.1.1. NIST dataset

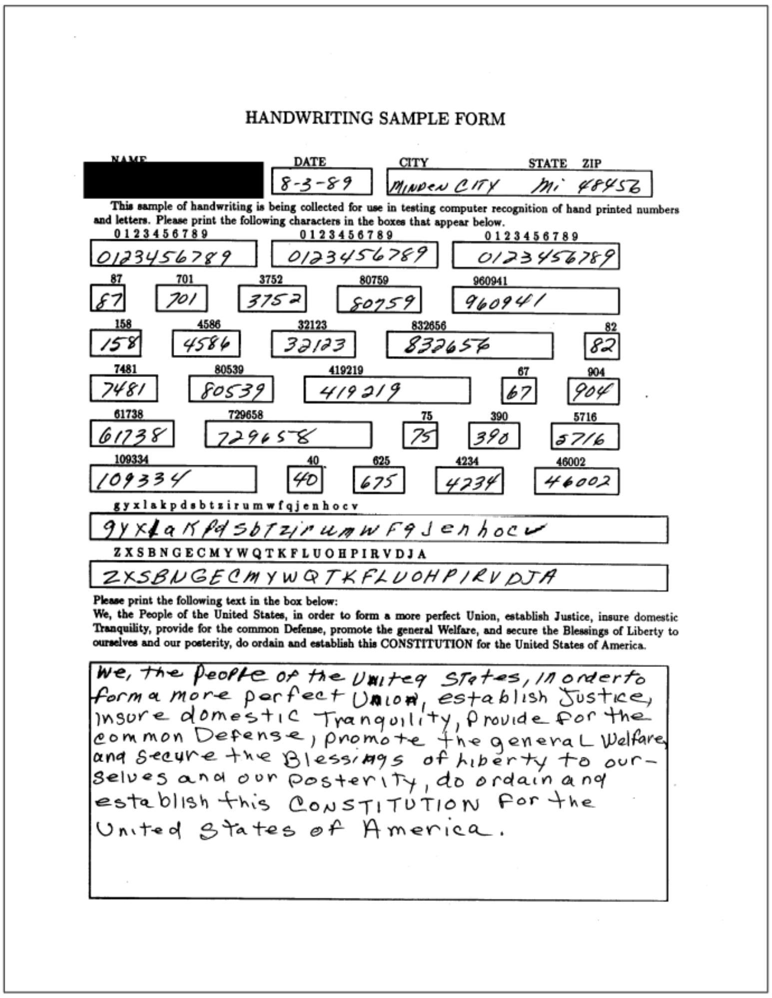
NIST Special Database 19 [12, 13] was first published by the American National Institute of Standards and Technology as CD-ROM in 1995, and then was re-released in 2016 using modern file formats. This offline database of handwritten digits and numerals contains 815255 segmented characters, each labeled with one of 62 classes (10 digits, 26 lowercase and 26 uppercase English alphabet characters). Those segmented characters are represented as monochromatic images, in resolution 128 by 128 pixels. Each of individual images were obtained from one of the 3669 completed forms (an example of one such is given in the Figure 1), where the segmentation of each of the images was checked manually. The database contains characters from 3596 authors. The database is provided through several hierarchies and it is suitable for the tasks of author identification, handwriting recognition, etc. The authors also propose internal division of data for training and testing (recommended data for testing comes from high school students' handwriting).

2.1.2. MNIST dataset

The MNIST dataset of offline handwritten digits, developed by Le Cun et all. in 1998 [8], has become one of the most famous and the most important datasets in machine learning, classification and computer vision tasks. MNIST was the first famous dataset derived from the larger NIST Special Database 19. The dataset contains 60000 images in the training set and 10000 images in the test set, labeled with one out of ten possible digits. The images are in grayscale, measuring 28 by 28 pixels. The images of the MNIST training and test set were created uniformly from the NIST training and test set, by selecting 50% of the images from each, while maintaining the identical distribution of the MNIST training and test sets. Over the years, this dataset has been widely used both in a large number of digit recognition systems and as a core dataset when introducing basic machine learning and pattern recognition concepts.

2.1.3. EMNIST dataset

The Extended MNIST (EMNIST) dataset represents a younger dataset created from the NIST database. It was created in 2017 by Cohen et al. [1]. It consists of several sub-datasets: EMNIST By_Class and



HANDWRITING SAMPLE FORM

[redacted] DATE CITY STATE ZIP

8-3-89 MINDEN CITY MI 48456

This sample of handwriting is being collected for use in testing computer recognition of hand printed numbers and letters. Please print the following characters in the boxes that appear below.

0123456789 0123456789 0123456789

0123456789 0123456789 0123456789

87 701 3752 80759 960941

87 701 3752 80759 960941

158 4586 32123 832656 82

158 4586 32123 832656 82

7481 80539 419219 67 904

7481 80539 419219 67 904

61738 729658 75 390 5716

61738 729658 75 390 5716

109334 40 625 4234 46002

109334 40 675 4234 46002

gyxlakpdsbtzirumwfqjenhocv

gyxlakpdsbtzirumwfqjenhocv

ZXSBNGECMYWQTKFLUOHPIRVDJA

ZXSBNGECMYWQTKFLUOHPIRVDJA

Please print the following text in the box below:
We, the People of the United States, in order to form a more perfect Union, establish Justice, insure domestic Tranquility, provide for the common Defense, promote the general Welfare, and secure the Blessings of Liberty to ourselves and our posterity, do ordain and establish this CONSTITUTION for the United States of America.

We, the People of the Uniteq States, inorderto form a more perfect Union, establish Justice, insure domestic Tranquility, Provide for the common Defense, promote the general Welfare, and secure the Blessings of Liberty to ourselves and our posterity, do ordain and establish this CONSTITUTION for the United States of America.

Figure 1. One of the NIST data gathering forms [9]

EMNIST By_Merge Datasets, EMNIST Balanced Dataset, EMNIST Letters Dataset and EMNIST MNIST Dataset. The images in each of them are monochromatic and are in resolution 28 by 28 pixels, while depending on the dataset, the labels to which they are marked differ. The EMNIST By_Class and By_Merge datasets both contains all NIST images, but differ in number of labels and image distributions in each label. By_Class dataset contains images labeled with 62 classes (NIST like), while in By_Merge dataset the upper and lower case examples for the classes C I J K L M O P S U V W X Y Z have been merged. EMNIST Balanced dataset consists of 131600 images labeled with one of the 47 classes. This dataset is created with intent to avoid misclassification errors caused by lower and upper versions of the same character. EMNIST Letters dataset consists of images labeled with one of the 26 English alphabet classes (uppercase and lowercase letters are merged like some classes in previous sub-dataset). EMNIST MNIST dataset consists of 280000 MNIST-like images, labeled with 10 digit classes.

The same conversion process was used when creating each of the sub-datasets. The authors simulated the conversion process that was used in creating the MNIST dataset, in order to share the same structure. The process of converting NIST images involves sequentially adding Gaussian noise (with the standard deviation parameter set to 1), extracting regions around a character, character centering, resizing, and resampling to obtain an image with the appropriate MNIST-like dimensions.

The impact of the MNIST dataset is also evident in the fact that there are datasets in other areas (such as computer vision) that follows the same structure and are named like this viral dataset [27]. In this paper we introduce MNIST like dataset, but doubled in number of samples and in resolution. The created dataset, which we called *DoubledMNIST*, is fully described in Section 3.

2.2. Genetic CNN

With the rise in number of layers in the CNNs, deeper neural networks are more difficult to train, and using skip connections proved invaluable and allowed for easier training of substantially deeper networks [6]. Those skip connections enable identity functions to be learned easily where needed. Those skip connections can be manually selected, but since the number of possible skip connections grows exponentially, and because evaluating each model can take a long time, in practice it is impossible to try each possible architecture. Many handcrafted CNN architectures exist, but since manually searching the space of all architectures is impractical, it gives a great incentive for automatic search for a good architecture.

A possible solution for finding a good architecture automatically is by using metaheuristic approach. In [4] authors proposes an encoding method of representing each network structure as a fixed length binary string. They define genetic operations: selection, mutation, and crossover to generate better suited individuals and eliminate weak ones. The fitness of an individual is defined through as their recognition accuracy, which is gathered through evaluation on a given reference dataset. The obtained structures are transferrable to other datasets image-based datasets.

Another approach focuses on automatically constructing CNN architectures for an image classification task based on Cartesian genetic programming (CGP). They use convolutional blocks and tensor concatenation as the node functions in CGP. Their results are comparable with handcrafted state-of-the-art models [5].

3. DoubledMNIST Dataset

After developing an architecture that solves the classification task on MNIST dataset, the natural next step of learning is to modify it to solve classification tasks on similar datasets. This was one of the basic motives that inspired us to create this kind of dataset. In addition, we aimed to create a dataset that is simple in structure (like a MNIST dataset) but still has more demanding dimensions, which would highlight all the advantages and disadvantages of various classification techniques that can be applied to it, which on the other hand would be a suitable dataset for an introduction to machine learning. On the other hand, the dimensions of the created dataset as well as its structure and complexity qualify it for more advanced applications than just educational ones. One of the reasons why the MNIST dataset has become very popular was its dimensions, which are extremely suitable for testing different types of deep learning architectures [27]. Our idea was to create a dataset that would serve as the second phase of testing in the development of those architectures, which, after the MNIST, would have to satisfy the more difficult requirements of our dataset.

The created dataset consists of 140000 images, 120000 in the training set and 20000 in the test set. All images are monochromatic, measuring 56 by 56 pixels. Each image, as in the MNIST dataset, is labeled with one out of ten labels (digits from 0 to 9). A random sample of our database images is given in the Figure 2.

Each image in our dataset was created by processing images of the NIST database, mostly following the instructions given in [1], which in turn follow the same conversion process described in [8]. For each of the ten digits, our conversion process first randomly selects 14000 images corresponding to it from the NIST database (12000 for a DoubledMNIST training set and 2000 for a DoubledMNIST test set), which we later transform from pixel binary images in resolution 128 by 128 to 8-bit gray-scale images in resolution 56 by 56. We used a random selection of which of the images from the NIST database would be in our dataset to avoid possible bias to individual writers' groups and their handwriting.

After loading the chosen image from the NIST dataset, the character itself is first cropped and then borders padded using a two pixel padding. We have taken the actual value of the offset from the mentioned publication, and their conversion process [1]. Thereafter, to soften the character borders, we blurred cropped image using a Gaussian filter with standard deviation set to 2. A border frame is then placed around a character, thus completing the extraction. It is important to emphasize here that the dimensions of the border frames differ greatly, since the dimension of the character is certainly one of the characteristics of the manuscript. Following the principles used in [1], we tried to use the maximum amount of space available, and therefore we did not downsample the cropped characters into smaller resolutions before the final one. Instead we then convert the cropped image of dimensions $height \times width$

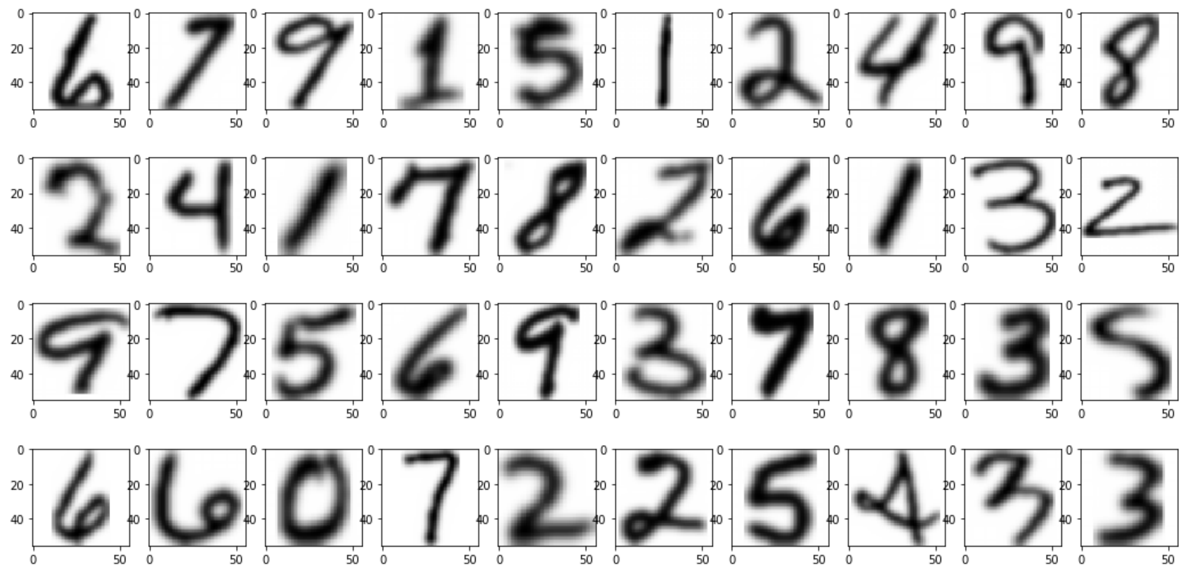


Figure 2. Sample of images from the DoubledMNIST database.

into a square image of dimensions *max(height, width)* $\times$ *max(height, width)* by extending the shorter dimension with the blank pixels while keeping the character centered in the image.

The next step was to resize an image of dimensions *max(height, width)* $\times$ *max(height, width)* into a dimensions 56×56. We did this downsampling (eventually upsampling) using bi-cubic interpolation. Finally, the image pixels are scaled into the 8-bit range. An example of the conversion described above is given in Figure 3.

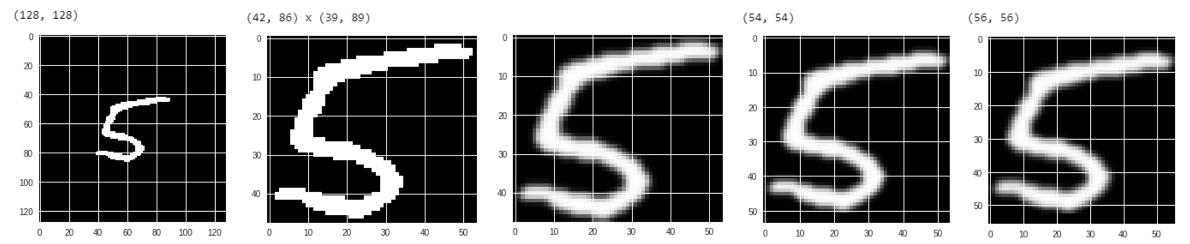


Figure 3. Conversion process: (1) original 128×128 NIST binary images; (2) cropped character with padding set to 2; above the image we can see pixel indices that represent the boundaries of our character in the original image; (3) character after applying Gaussian filter with sigma set to 2; (4) image centered into a squared *max(height, width)* $\times$ *max(height, width)* frame; (5) image sampled in resolution 56×56 using bi-cubic interpolation

Additionally, while creating this dataset, we tried various combinations of noise addition and interpolation. Some of the variations we applied were: used Gaussian noise and Bicubic interpolation, Gaussian noise with Lanczos interpolation and Bilinear interpolation, Bicubic interpolation with Gaussian Laplace filter and Fourier Gaussian filter and so on. We evaluated the quality of each combination by training a simple model of K Nearest Neighbors classifier in the images modified by that combination. We chose the first variant, just like in the paper [1], but we used the standard deviation set to 2 instead of 1 as much as they used in the mentioned paper (since such a filter achieved better results through our evaluations).

4. Network representation

Many network architectures, such as [6, 7], can be partitioned into stages. In each stage the width, height and depth of the data cube remain unchanged. Adjacent stages are connected with pooling layers which reduce both width and height of the data. The polling usually performs 2×2 max polling operation, which halves the width and height by transforming each 2×2 pixel group into one pixel whose value is the maximum of the pixel group. The number of channels (filters) within convolutions of the same stage is the same.

That observation is leveraged to define a set of networks which can be encoded into a binary string of fixed length. Because of that, in our work we used the binary string network representation which was proposed in [4]. In our scenario, a CNN is composed of S stages, each containing K_i nodes $(i = 1, ..., S)$. The nodes within a stage are ordered, and connections are only allowed from a lower number towards a higher numbered node. Each node represents a convolutional operation. If a node has multiple inputs, then they are all summed up element-wise [4]. After convolution, the ReLU activation is performed. The fully connected end of the network is not encoded, as its hyperparameters are static. Since each node can be an input only to higher numbered nodes, that means that $(K_i - 1) + (K_i - 2) + ... + 2 + 1 = \frac{K_i(K_i-1)}{2}$ bits are needed to encode a stage containing K_i nodes.

Some encodings would represent invalid networks, thus default nodes are introduced in each stage. They are the first and the last nodes in the stage, and since they are always present, they are not encoded in the binary string.

There are a few special cases:

- If a node does not have any input or output, then it is removed.
- If a node does not have an input, then we connect the first node in the stage to it.
- If a node does not have an output, then we connect it to the last node in the stage.

We have adapted our genetic algorithm so that the chromosome is represented by a binary string representing the architecture of the convolutional part of the neural network, which we defined as described above.

5. Method

The core idea of the genetic algorithm is to obtain a good solution to a problem by generating increasingly better solutions through the process of evolution. Network architecture is encoded as a binary string of fixed length. That string represents a gene of an individual. Individuals from the population have a higher chance to pass on their genes to the next generation if they are more fit for a given task. Through many generations its expected to arrive to a population that has many good individuals and the best individual out of that last generation represents the solution.

The evolution process consists of selection, mutation, and crossover. The selection process allows stronger individuals to be preserved, and for weaker individuals to be eliminated. Crossover process combines genes of two individuals to create individuals for the new generation. Mutation randomly changes a gene of an individual, thus introducing more variety within a population.

We denote the n-th individual in generation t as $M_{t,n}$ and fitness of that individual as $f_{t,i}$.

Algorithm 1: Genetic algorithm for generating the appropriate network architecture

Input: the testing dataset T, number of generations G, number of individuals in each generation N, mutation and crossover probabilities p_M and p_C, mutation parameter q_M and crossover parameter q_C.

Initialization: generate randomized individuals, train them, and compute their fitness (evaluate classification accuracy);

for t = *1,...,G* **do**

- **Selection:** perform selection using a roulette method to p;
- **Crossover:** perform crossover with probability p_C and crossover parameter q_C;
- **Mutation:** perform mutation on individuals which have not had crossover with probability p_M and mutation parameter q_M;
- **Construction:** construct CNN from the gene encoding it;
- **Inheritance:** inherit the stage weights from the most similar individual of the last generation;
- **Training:** train the constructed networks, where number of epochs depends on number of inherited stages;
- **Evaluation:** evaluate all individuals to get their fitness;

end

Output: individuals of the final generation and their classification accuracy.

5.1. Initialization phase

The initial generation of individuals is generated by assigning each bit in the binary string a random value from Bernoulli distribution $\mathcal{B}(0.5)$ as described in [4]. Then all individuals from the initial generation are fully trained on the training dataset and evaluated on a testing dataset to get their fitness. After that, the genetic evolution process is repeated for a set number of generations.

5.2. Selection phase

Here we use roulette selection, where the least fit individual is always eliminated. In roulette selection, each individual has a chance to pass on its genes to the next generation, where the probability of that event is proportional to the individuals fitness in comparison to the fitness of all other individuals. The sampling is performed N times (number of individuals in each generation) with replacement, thus each individual may be selected multiple times. The probability of individual $M_{t,n}$ passing the selection is equal to $f_{t,i}/\sum_{i=1}^{N}(f_{t,i} - f_{t,min})$, where $f_{t,min} = \min_{i=1}^{N} f_{t,i}$.

5.3. Crossover phase

The crossover process combines the genes of two individuals to create one or two new individuals. Here crossover of two individuals always results in two new individuals. When two individuals are in a crossover they swap a whole stage. In that way the learned connections within a stage are kept through the generations, while still introducing more variety in individuals. Candidates for crossover are pairs of individuals $(M_{t,2i-1}, M_{t,2i}), \forall i = 1, .., \lfloor N/2 \rfloor$ [4]. The probability of crossover between two individuals is p_C, and the probability of two stages being swapped is q_C.

5.4. Mutation phase

Mutation can occur only if an individual did not go through the crossover process. In that case the individual starts going through mutation with probability p_M. Then each bit in the individuals string representation has a low chance of being inverted, defined in q_M [4]. Mutation ensures additional variety in individuals and allows for exploring different architectures within each stage.

5.5. Construction phase

The binary encoded string of each individual is parsed and the graph is constructed, where graph nodes represent the convolution operations. Connections within graph nodes signal that there is a connection between those two nodes. The whole CNN is constructed by parsing the binary string according to the representation discussed in Section 4. The end of the network always consists of a flatten layer, followed by a dense (fully connected) layer with 32 units, and finally a dense layer with softmax activation and number of units equal to the number of possible classes.

5.6. Training phase

Training is performed on a constructed CNN model for a set amount of epochs. The more stages were inherited in inheritance phase, the less number of epochs is needed to train the model. Training is done on the training dataset which is the same for all individuals.

5.7. Evaluation phase

Evaluation phase is performed to get the fitness of an individual. The trained model is evaluated on a testing dataset to get its classification accuracy, which is used as its fitness.

6. Evaluation

In this section we will describe the results we achieved using proposed method. We evaluated our model on the MNIST dataset [8]. In addition, we trained the learned architecture on the created dataset DoubledMNIST, thus defining the first classification results on it. As hardware resources, we used the Google Colab environment, a free cloud service hosted by Google Inc. Additionally, all of our source code is publicly available at the web address *github.com/MilanCugur/Genetic_Evolution_For_CNN.*

The choice of architecture was made on the MNIST dataset, given its characteristics. When choosing the right architecture, three convolutional stages with 3, 4 and 5 convolutions were used, respectively. Previously mentioned architectures are represented by the genome with a length of 19. The number of filters is the same for each convolution within the same stage, and is equal to 32, 48 and 64 respectively across the stages. In our implementation, similar to [4], we used $p_M = 0.8$ and $q_M = 0.1$ for the mutation parameter values, while we fixed $p_C = 0.2$ and $q_C = 0.3$ for the values of the crossing parameters.

When evaluating the proposed method, we created several different benchmarks that cover several basic evolution scenarios. In each of them, the proposed method gave a significant improvement. Specific scenarios and results are given with:

- 4 individuals trained over 4 generations: baseline training time: 32m 44s, proposed method training time: 18m 01 s
- 20 individuals trained over 20 generations: baseline training time: 2h 33m 54s, proposed method training time: 1h 06m 25 s
- 2 individuals trained over 20 generations: baseline training time: 25m 25s, proposed method training time: 18m 18 s
- 20 individuals trained over 2 generations: baseline training time: 1h 22m 22s, proposed method training time: 46m 55 s

In the previous list, the first two points correspond to the scenario when we have an identical (in the first point insufficient, but in the second point sufficient) number of generations and individuals in them. In the second point, only one sixth of the MNIST dataset was used because of the computational complexity and limitations of our hardware. The third point simulates an evolution scenario when we have enough individuals, however a very modest number of generations, while the last point simulates the opposite situation when the number of generations is sufficient, but when we do not have enough individuals in them. It is important to point out that in each of the four preceding scenarios, our represented method has yielded a significant improvement measured in time savings, without losing out on the quality of the generated architectures.

The standard evolution scenario when we have enough (10) individuals in each of the 10 generations of evolution (trained across the entire MNIST dataset) generated the architectures as in Figure 4 (In Appendix A, a better-resolution image of generated architectures are provided), while the results of the two best individuals of each generation can be seen in Figure 5.

The first architecture of Figure 4 was trained on a DoubledMNIST dataset with training lasting about 30min and stopped with an early stop technique after 14 epochs, with 99.495% precision on the test set. This sets the results of the classification on the created dataset.

7. Conclusion

In this paper we showed that it is possible to accelerate the genetic algorithm for training deep convolutional neural networks if individuals of one generation use the knowledge (specifically parameters)

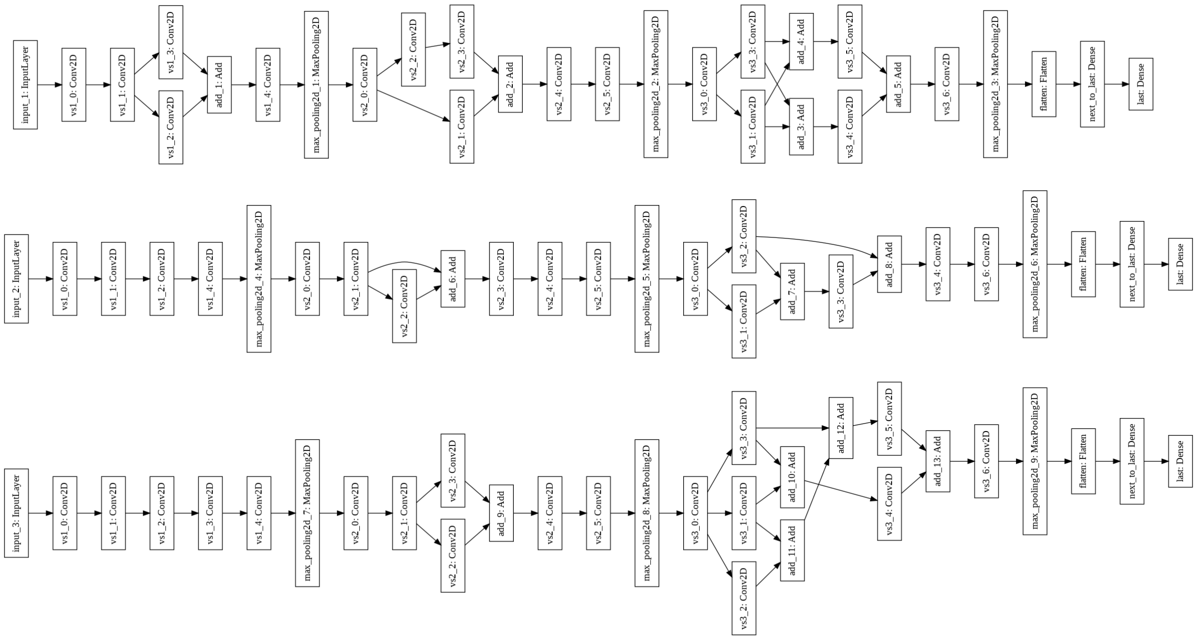


Figure 4. Generated architectures using presented method on NIST dataset

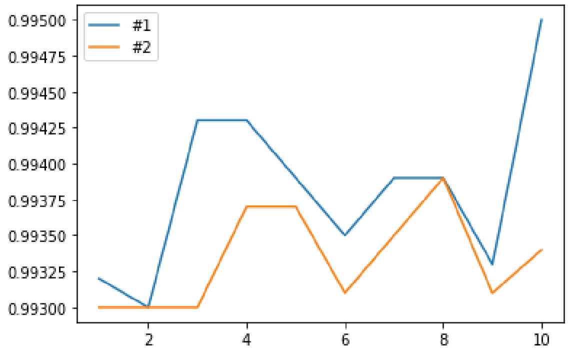


Figure 5. Top two individuals' precision through 10 genetic algorithm training epochs

learned by their ancestors. By including such parent-child connections, we have made it possible to greatly reduce the execution time of the algorithm itself, given that most of the execution time in genetic algorithms of this type is spent in training the individuals of each generation. Additionally, the reduced training we perform in the case of inheriting a certain number of first layers of the network from one parent contributes to the robustness of the training process itself as the first layers of the final network (which define the new attributes which will be fed into the deeper layers of the network) are retrained through multiple epochs (however many they survive).

In addition, the tests we performed showed that our system behaves extremely well under various evolutional scenarios: when we have enough time and individuals, in situations where the number of generations is limited for some reason, but we have a population which is large enough, as well as in the situation when we have enough time for evolution but the number of individuals in the population is limited.

Another contribution of this paper is the creation of a new dataset of offline handwritten English alphabet characters that should be heir to the popular MNIST dataset, given its dimensions and characteristics. The created dataset is ideal for teaching or learning ML, e.g. when architectures created on MNIST are adapted to the more serious requirements called for by our new dataset. The results we achieved by training the architecture obtained by executing the genetic algorithm on the MNIST dataset are the first ones on the DoubledMNIST dataset.

Possible directions for improvement of our work are related to the introduction of stronger ancestor-descendant relationships, that is, the inheritance of more information that ancestors have learned, possibly combining knowledge from multiple ancestors at the same time within the same individual.

References

[1] **Cohen, G., Afshar, S., Tapson, J., van Schaik, A.** EMNIST: an extension of MNIST to handwritten letters. *arXiv preprint arXiv:1702.05373.*, 2017.

[2] **Rikhtegar, A., Pooyan, M., Manzuri-Shalmani, M. T.** Genetic algorithm-optimised structure of convolutional neural network for face recognition applications. *IET Computer Vision, 10(6), 559-566.*, 2016.

[3] **Voß, S., Martello, S., Osman, I. H., Roucairol, C. (Eds.).** Meta-heuristics: Advances and trends in local search paradigms for optimization. *Springer Science and Business Media.*, 2012.

[4] **Xie, L., Yuille, A.** Genetic cnn. *In Proceedings of the IEEE International Conference on Computer Vision (pp. 1379-1388).*, 2017.

[5] **Suganuma, M., Shirakawa, S., Nagao T.** A genetic programming approach to designing convolutional neural network architectures. *In Proceedings of the Genetic and Evolutionary Computation Conference (pp. 497-504).*, 2017.

[6] **He, K., Zhang, X., Ren S., Sun J.** Deep Residual Learning for Image Recognition. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[7] **Simonyan K., Zisserman A.** Very Deep Convolutional Networks for Large-Scale Image Recognition. *International Conference on Learning Representations*, 2014.

[8] **LeCun Y., Bottou L., Benigo Y., Haffner P.** Gradient-based learning applied to document recognition., *Proceedings of the IEEE, 86(11):2278-2324*, November 1998.

[9] **Hussain, R., Raza, A., Siddiqi, I., Khurshid, K., Djeddi, C.** A comprehensive survey of handwritten document benchmarks: structure, usage and evaluation. *EURASIP Journal on Image and Video Processing, 2015(1), 46.*, 2015.

[10] **Grosicki, E., Carre, M., Brodin, J. M., Geoffrois, E.** RIMES evaluation campaign for handwritten mail processing., 2008.

[11] **Marti, U. V., Bunke, H.** The IAM-database: an English sentence database for offline handwriting recognition. *International Journal on Document Analysis and Recognition, 5(1), 39-46.*, 2002.

[12] **Grother, P. J.** NIST special database 19. *Handprinted forms and characters database, National Institute of Standards and Technology.*, 1995.

[13] **Grother, P. J.** NIST Special Database 19. *NIST Handprinted Forms and Characters Database (No. World Wide Web-Internet and Web Information Systems).*, 2016.

[14] **Singh, S., Hewitt, M.** Cursive digit and character recognition in CEDAR database. *In Proceedings 15th International Conference on Pattern Recognition. ICPR-2000 (Vol. 2, pp. 569-572). IEEE.*, 2000.

[15] **Guyon, I., Schomaker, L., Plamondon, R., Liberman, M., Janet, S.** UNIPEN project of on-line data exchange and recognizer benchmarks. *In Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3-Conference C: Signal Processing (Cat. No. 94CH3440-5) (Vol. 2, pp. 29-33). IEEE.*, 1994.

[16] **Shivram, A., Ramaiah, C., Setlur, S., Govindaraju, V.** IBM_UB_1: A dual mode unconstrained English handwriting dataset. *In 2013 12th International Conference on Document Analysis and Recognition (pp. 13-17). IEEE.*, 2013.

[17] **Al Maadeed, S., Ayouby, W., Hassaïne, A., Aljaam, J. M.** Quwi: An arabic and english handwriting dataset for offline writer identification. *In 2012 International Conference on Frontiers in Handwriting Recognition (pp. 746-751). IEEE.*, 2012.

[18] **Kleber, F., Fiel, S., Diem, M., Sablatnig, R.** Cvl-database: An off-line database for writer retrieval, writer identification and word spotting. *In 2013 12th International Conference on Document Analysis and Recognition (pp. 560-564). IEEE.*, 2013.

[19] **Liu, C. L., Yin, F., Wang, D. H., Wang, Q. F.** CASIA online and offline Chinese handwriting databases. *In 2011 International Conference on Document Analysis and Recognition (pp. 37-41). IEEE.*, 2011.

[20] **Su, T., Zhang, T., Guan, D.** HIT-MW dataset for offline Chinese handwritten text recognition., 2006.

[21] **Liu, C. L., Yin, F., Wang, D. H., Wang, Q. F.** Online and offline handwritten Chinese character recognition: benchmarking on new databases. *Pattern Recognition, 46(1), 155-162.*, 2013.

[22] **Pechwitz, M., Maddouri, S. S., Märgner, V., Ellouze, N., Amiri, H.** IFN/ENIT-database of handwritten Arabic words. *In Proc. of CIFED (Vol. 2, pp. 127-136). Citeseer.*, 2002.

[23] **Märgner, V., El Abed, H.**ICDAR 2009 Arabic handwriting recognition competition. *In 2009 10th International Conference on Document Analysis and Recognition (pp. 1383-1387). IEEE.*, 2009.

[24] **Sarkar, R., Das, N., Basu, S., Kundu, M., Nasipuri, M., Basu, D. K.** CMATERdb1: a database of unconstrained handwritten Bangla and Bangla–English mixed script document image. *International Journal on Document Analysis and Recognition (IJDAR), 15(1), 71-83.*, 2012.

[25] **Chaudhuri, B. B.** A complete handwritten numeral database of Bangla–a major Indic script., 2006.

[26] **Djeddi, C., Gattal, A., Souici-Meslati, L., Siddiqi, I., Chibani, Y., El Abed, H.** LAMIS-MSHD: a multi-script offline handwriting database. *In 2014 14th International Conference on Frontiers in Handwriting Recognition (pp. 93-97). IEEE.*, 2014.

[27] **Xiao, H., Rasul, K., Vollgraf, R.** Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747.*, 2017.

[28] **Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A. A.** Inception-v4, inception-resnet and the impact of residual connections on learning. *In Thirty-First AAAI Conference on Artificial Intelligence.*, 2017.

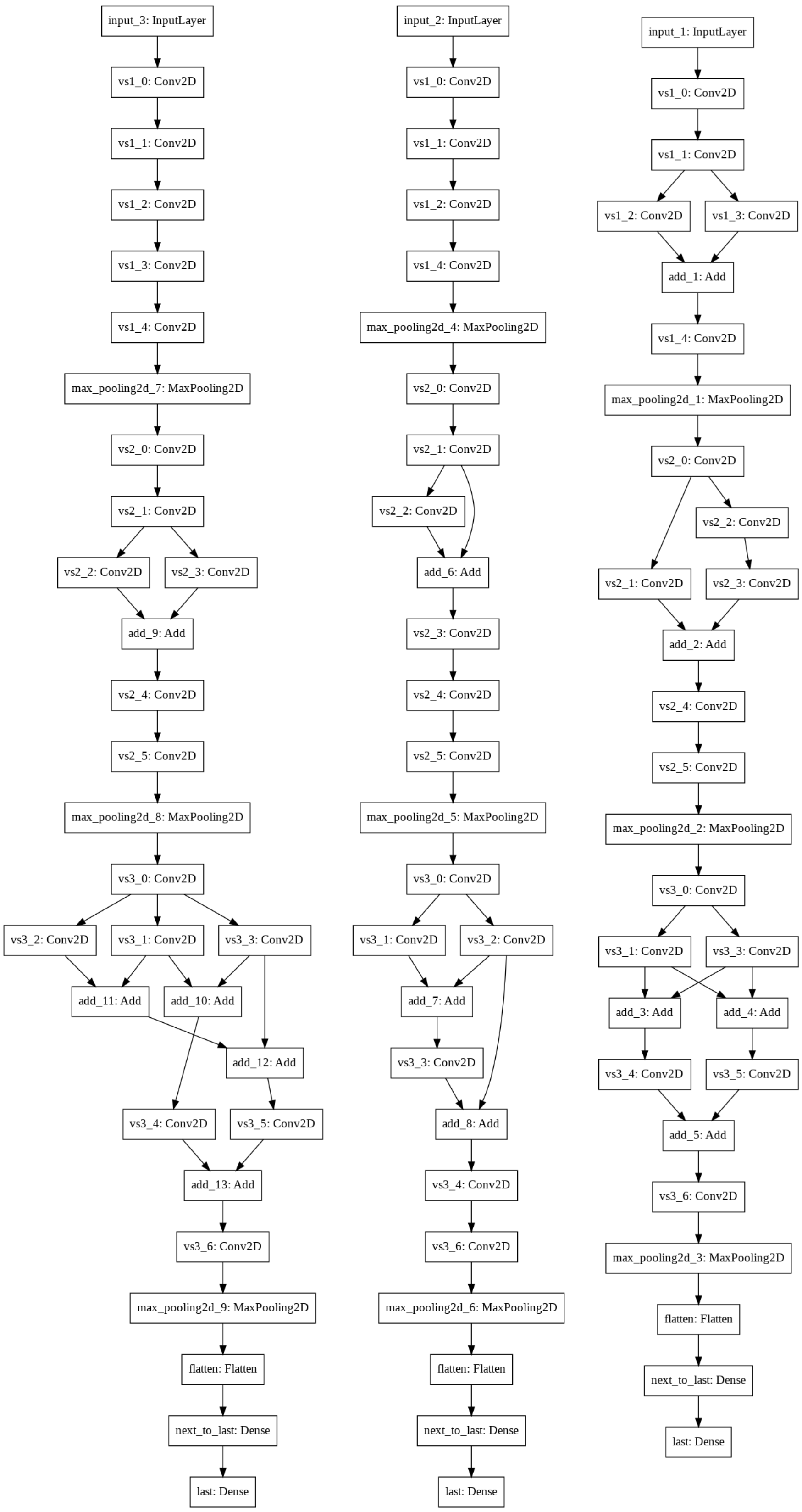
[29] **Wu, Z., Shen, C., Van Den Hengel, A.** Wider or deeper: Revisiting the resnet model for visual recognition. *Pattern Recognition, 90, 119-133.*, 2019.

[30] **Lu, Z., Whalen, I., Boddeti, V., Dhebar, Y., Deb, K., Goodman, E., Banzhaf, W.** NSGA-NET: a multi-objective genetic algorithm for neural architecture search. *arXiv preprint arXiv:1810.03522.*, 2018.

[31] **Suganuma, M., Shirakawa, S., Nagao, T.** A genetic programming approach to designing convolutional neural network architectures. *In Proceedings of the Genetic and Evolutionary Computation Conference (pp. 497-504). ACM.*, 2017.

[32] **Oullette, R., Browne, M., Hirasawa, K.** Genetic algorithm optimization of a convolutional neural network for autonomous crack detection. *In Proceedings of the 2004 congress on evolutionary computation (IEEE Cat. No. 04TH8753) (Vol. 1, pp. 516-521). IEEE.*, 2004.

Appendix A:


input_3: InputLayer
vs1_0: Conv2D
vs1_1: Conv2D
vs1_2: Conv2D
vs1_3: Conv2D
vs1_4: Conv2D
max_pooling2d_7: MaxPooling2D
vs2_0: Conv2D
vs2_1: Conv2D
vs2_2: Conv2D
vs2_3: Conv2D
add_9: Add
vs2_4: Conv2D
vs2_5: Conv2D
max_pooling2d_8: MaxPooling2D
vs3_0: Conv2D
vs3_2: Conv2D
vs3_1: Conv2D
vs3_3: Conv2D
add_11: Add
add_10: Add
add_12: Add
vs3_4: Conv2D
vs3_5: Conv2D
add_13: Add
vs3_6: Conv2D
max_pooling2d_9: MaxPooling2D
flatten: Flatten
next_to_last: Dense
last: Dense
input_2: InputLayer
vs1_0: Conv2D
vs1_1: Conv2D
vs1_2: Conv2D
vs1_4: Conv2D
max_pooling2d_4: MaxPooling2D
vs2_0: Conv2D
vs2_1: Conv2D
vs2_2: Conv2D
add_6: Add
vs2_3: Conv2D
vs2_4: Conv2D
vs2_5: Conv2D
max_pooling2d_5: MaxPooling2D
vs3_0: Conv2D
vs3_1: Conv2D
vs3_2: Conv2D
add_7: Add
vs3_3: Conv2D
add_8: Add
vs3_4: Conv2D
vs3_6: Conv2D
max_pooling2d_6: MaxPooling2D
flatten: Flatten
next_to_last: Dense
last: Dense
input_1: InputLayer
vs1_0: Conv2D
vs1_1: Conv2D
vs1_2: Conv2D
vs1_3: Conv2D
add_1: Add
vs1_4: Conv2D
max_pooling2d_1: MaxPooling2D
vs2_0: Conv2D
vs2_2: Conv2D
vs2_1: Conv2D
vs2_3: Conv2D
add_2: Add
vs2_4: Conv2D
vs2_5: Conv2D
max_pooling2d_2: MaxPooling2D
vs3_0: Conv2D
vs3_1: Conv2D
vs3_3: Conv2D
add_3: Add
add_4: Add
vs3_4: Conv2D
vs3_5: Conv2D
add_5: Add
vs3_6: Conv2D
max_pooling2d_3: MaxPooling2D
flatten: Flatten
next_to_last: Dense
last: Dense