Faculty of Electrical Engineering and Computer Science

University of Maribor

Implementing Evolutionary Neural Architecture Search (ENAS)

Documentation

Kacper Nowakowski

Maribor, 2022

Contents

1. Introduction

Deep Neural Networks (DNNs), as the cornerstone of deep learning, have demonstrated their great success in diverse real-world applications, including image classification, natural language processing, speech recognition, to name a few. Generally, the performance of DNNs depends on two aspects: their architectures and the associated weights. The optimal weights are often obtained through a learning process. When the termination condition is satisfied, which is commonly a maximal iteration number, the algorithm can often find a good set of weights. This kind of process has been very popular owing to its effectiveness in practice, and has become the dominant practice for weight optimization, although they are principally local-search algorithms.

On the other hand, obtaining the optimal architectures cannot be directly formulated by a continuous function, and there is even no explicit function to measure the process for finding optimal architectures. To this end, there has been a long time that the promising architectures of DNNs are manually designed with rich expertise. This can be evidenced from the state of the arts, such as VGG, ResNet and DenseNet. However, such a design process is labour intensive because of the trial-and-error process, and also not easy to realize due to the rare expertise in practice. Moreover, DNN architectures are often problem-dependent. If the distribution of the data is changed, the architectures must be redesigned accordingly.

But what, if we could go one step further and it did not required that much of expertise and effort? Neural Architecture Search (NAS), which aims to automate the architecture designs of deep neural networks, is identified as a promising answer to that question. Among different methods to realize NAS, Evolutionary Computation (EC) methods have recently gained much attention and success. That is why this work focuses on analyzing implementation of the combination of these algorithms which is called ENAS.

Most of the information for this work was received from Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, K. C. Tan "A Survey on Evolutionary Neural Architecture Search" 2020.

2. Evolutionary Algorithm

2.1. Defining an individual

Since the main idea is to use EC in that solution and EC algorithms are based on population that consists of many individuals, the very first step is to define how individual will look like in that case. In other words, it is required to parameterize such abstract thing as ANN so it is possible to store it as a vector with EC algorithms can work with. This part is very important as it determines the search space i.e., architectures that this algorithm will be able to achieve. It means that with more aspects of an ANN encoded more diverse results can be obtained thus chance to get global optimum increases.

The most basic problem that needs to be solved is to parameterize layers somehow. Based on the options mentioned in the article block-based encoding space was chosen where multiple layers are combined into blocks, forming repeating patterns with parameters special for every single block. (Fig. 2.1) Cell-based encoding, which was other option, is unnecessarily restrictive as it uses only one type of block for the whole architecture. On the other hand layer-based encoding (which was yet another option) is far more difficult to implement. In addition, potential greater diversity is misleading, because many possible connection between layers is pointless and so discovering them using EA.

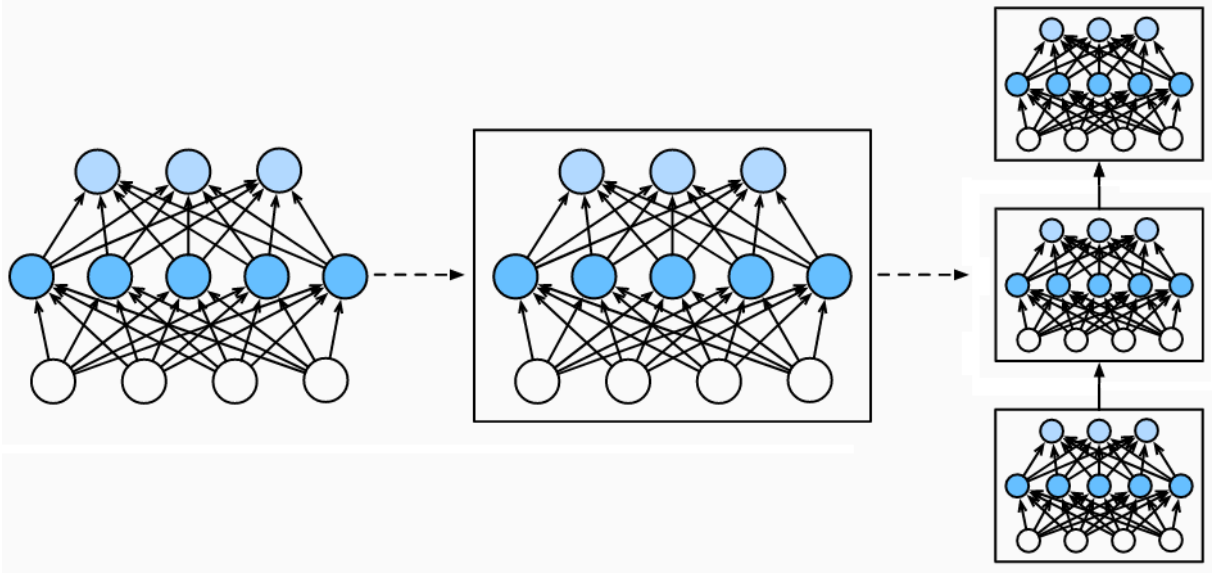


Figure 2.1: In block-based encoding multiple layers are combined into blocks, forming repeating patterns of larger models.

The great advantage of the block-based encoding space is the fact that there already are many traditional blocks and it is very easy to add them (or custom) to the existing code. Due to available computing capability only the most basic block for image classification was implemented, which is VGG block presented in Fig. 2.2. Parameters that are specified, are listed in Table **??**. Head of the every architecture is the same - dropout layer followed by typical dense layer with softmax activation function which is summarized in Fig. 2.3.

Table 2.1: Parameters defining architecture

Item	Parameters
Convolution layer	Feature size, kernel size, activation function
Pooling layer	Pool size, pooling type (max/avg)
VGG block	Convolution layers, pooling layer
Network	Blocks

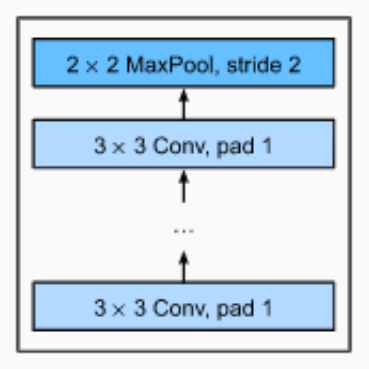


Figure 2.2: VGG Block

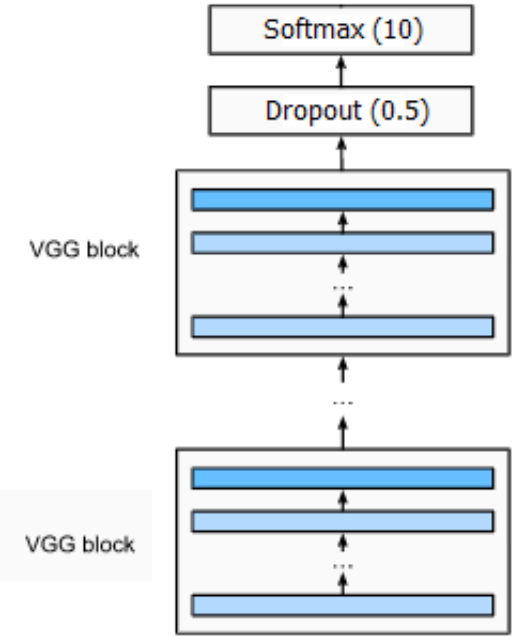


Figure 2.3: Schema of architectures that will be created by the script

2.2. Creating population

Creating population requires 2 parameters: population size and maximum amount of blocks which architecture can consists of. Generating population is totally random as script assigns one of possible value to each parameter, number of blocks and layers in blocks included.

2.3. Fitness evaluation process

For evaluation, used dataset is split into 3 parts - training, validation and testing dataset. Training dataset is obviously used for training every architecture. Validation dataset is used by script to evaluate trained architecture and result will be a part of fitness value "seen" by the algorithm. Testing dataset is used similarly as a validation set as it evaluates specimen but it is not "seen" by the script. It is required as EA can overfit to the validation set through the searching process.

But accuracy on dataset is only a part of the fitness. Taking into consideration that computing time is also important fitness is affected by the number of trainable parameters in architecture. As a consequence if architecture is only slightly better and much more bigger it will receive lesser grade.

It can occur that invalid architecture is created. It is possible because of the fact that feature size shrinks as it goes through the network. Then such a network gets 0 (worst mark) as a fitness value and should quickly disappear thanks to following operations.

Script saves some (it can be specified) best individuals that has the best validation fitness and the same number of architectures that got the best testing fitness. It will be used after searching process to analyze whether EA gave overfitted result or not.

2.4. Selection

Due to the great popularity in such cases tournament selection was chosen. It is quite simple algorithm to implement and removes the worst individuals gradually which is a good compromise between exploration and exploitation.

2.5. Crossover

Crossover is the first typical evolutionary operation. It combines two individuals with hope that good patterns that will be mixed will perform better together. With block-base encoding it is quite simple - one-point crossover between list of blocks. Even with different length it is not a problem, because it will generate two different size offsprings. However, the problem is with one-block sized architectures, since they cannot be split. Thus they are transferred further directly i.e. without crossover. The rest is paired randomly, but better individuals have more chance to be selected for reproduction. It is possible to pick two the same parents - it will just copy them.

2.6. Mutation

Next evolutionary operation is mutation. It is controlled by 3 parameters - mutation probability, block-size change probability and network-size change probability. The first one specifies chances for changing simple parameters like activation function or layer size. Former parameters relate to chance for changing amount of layers in one block and amount of blocks in the architecture. Therefore as a result of mutation number of layer as number of blocks can change. Last 2 parameters should not be very huge as they change individual drastically.

2.7. Succession

After re-evaluation of the new specimens elitism with elite size k=1 is used to replace the old population. It means that the best individual from old population will stay and the rest will be altered with newly generated architectures except the worst one. Thanks to this option even if

evolutionary operations fail to find better specimen, the next cycle will have access to the best individual so far so it can still be used.

3. Evaluation of the algorithm

3.1. Choosing dataset

In the experiment, ENAS was used for image classification on Fashion MNIST dataset with sample pictures presented in Fig. 3.1. The resolution of these pictures is 28x28 pixels, they are monochromatic and they are split into 10 classes. Unfortunately, it was the best possible option because of the modest computing capability. Whole process would take many hours, if greater or colorful pictures had been chosen.

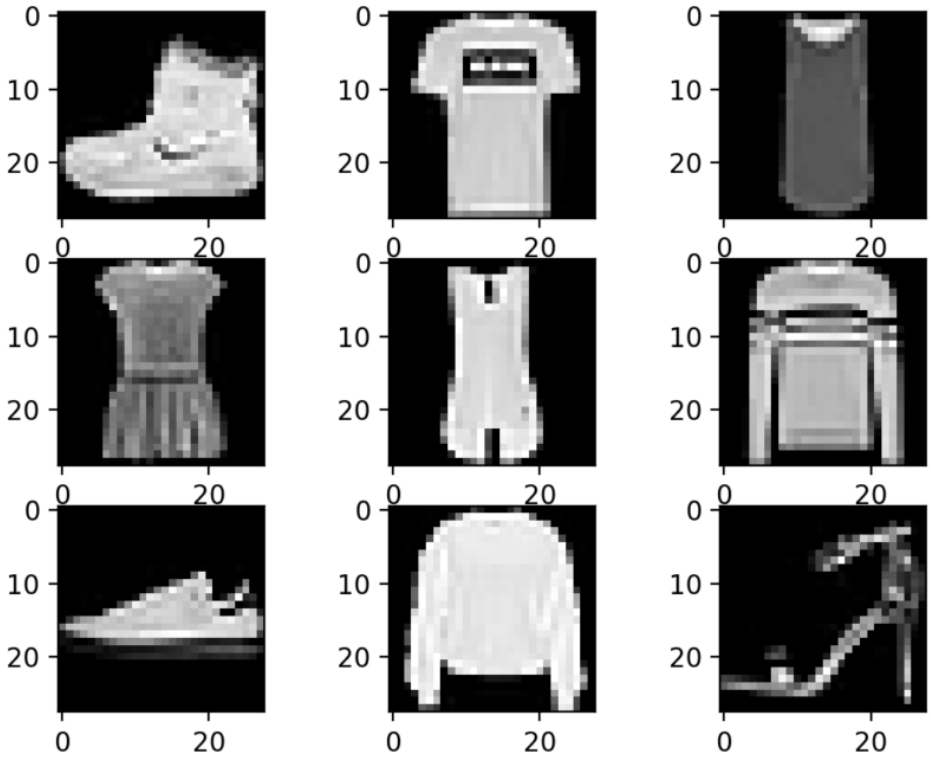


Figure 3.1: Sample Fashion MNIST pictures

3.2. Splitting dataset

Thanks to Keras library this dataset is already split into training and testing sets, consisting of 30,000 and 10,000 pictures respectively. It also guarantees similar distribution of classes. Training set is then split into the real training set and validation set with 0.2 ratio which gives 24,000 pictures for training and 6,000 for validating.

3.3. Setting up the parameters

Whole script was made in Jupyter Notebook environment. Functions directly related to ANN and dataset itself were taken from Keras library. Loss function used was categorical cross entropy and "adam" was the optimizer. Setting of every parameter in experiment is listed in the Table **??**.

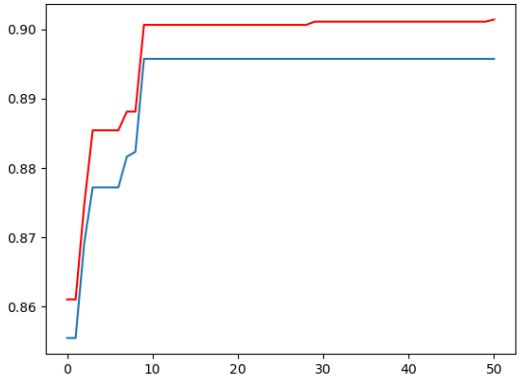
Table 3.1: Setting of the parameters used in the experiment

Parameter	**Options/Max value**
Feature size	(2, 4, 6, 10, 14, 22, 30, 46, 62)
Max kernel\pool size	5
Max convolutional layers in VGG block	3
Max block amount	3
Activation functions	(relu, sigmoid, tanh)
Mutation probability	50%
Block length change probability	40%
Network length change probability	30%
Bach size	128
Epochs	5
Population size	20
Iterations	50

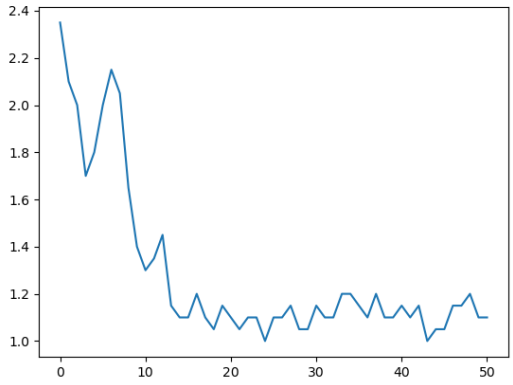
3.4. Results and observations

Diagram in Fig. 3.2a shows that the ENAS algorithm behave typically for the EA. At the very beginning it finds better solutions very quickly. However this speed diminish over 10 iterations and almost stops till the very end. Based on the fixed difference between the results we can assume that there is no significant overfitting. Final architecture has an accuracy of 89.5%.

Second diagram shows that these results couldn't occurred by an accident as we can clearly see declining trend. It is expected as state-of-art architecture consists of only one block.



(a) The best accuracy in following iterations. Red = validation set, Blue = testing set



(b) Average amount of blocks in the population

Figure 3.2

4. Conclusions

Based on the results it can be said that this implementation of ENAS is working properly even though it hadn't achieved the state-of-art accuracy, which is somewhere around 99%. In the end, basing on the diagrams, it was going in the right direction. Probably if DNNs had been training with 15 epochs and if there had been more iterations, it would have performed better. But it would have taken probably 20+ hours with available computing power. And that is the most important conclusion: ENAS is a really promising idea, but we are not ready for that yet. At least not when it comes for image classification problem. Even though I had very limited capabilities, this dataset is one of the simplest to classify. So it is not wrong to assume that it would required much advanced hardware and much more time for colored pictures with high resolution. Not everybody interested in ENAS can afford that. That is why many researches currently focus on how to optimize training process. According to the article there are some ways for architectures to remember what they already learnt before evolutionary operations. It is also possible to make algorithm to omit discovering architectures that are similar to the already discovered so it doesn't waste time.

It appears that now ENAS to be used effectively requires many special software additions that would reduce time needed. However, considering that ENAS algorithm should save time in comparison to manually creating architecture, it doesn't seem to be quite useful yet. Expertise seems to be good enough for now. But just for now. It is well known that average computing capabilities grow higher. And as new services requires more and more advanced algorithms, humans will not be able to create effective very deep neural networks, therefore we will have to pass the torch to the AI algorithms such as ENAS.