Computational Intelligence ( )

Submission Assignment 5

Instructor: JakubM. Tomczak *Name:* Elizabeth Dwenger, *Netid:* edr201

1 Introduction

Within the field of computational intelligence, evolutionary algorithms and neural networks are two frameworks which have been created to advance the goal of optimization. Evolutionary algorithms at their core are algorithms which have structures that emulate nature; similarly, a neural network approximates how the human brain is set up with the goal of finding patterns. The aim of this project is to combine these two structures by using an evolutionary algorithm to select the architecture of a neural network. Then, this architecture will be compared against the performance of two other algorithms, namely a fully connected neural network, and a convolutional neural network. The specific dataset this algorithm will be trained on is the Scikit-Learn Digits dataset, with the goal of finding the best performing algorithm to test on the test set.

2 Problem statement

This project is intertwined between three subgoals. These goals include implementing a convolutional neural network, implementing an evolutionary algorithm for neural architecture search, and evaluating the performance of the algorithm that is created with the aforementioned aspects. This leads to a focus on hyperparameter optimization within the neural architecture search. The aim of the model is to minimize the classification error of images by the neural network. Here, the classification error is measured using the Scikit-Learn Digits dataset.

2.1 Objective Function

The objective function in this project is the classification error that has been modified with the addition of a penalty. The penalty is in relation to the number of weights that are used in the convolutional neural network, and it is calculated by

$$Objective = ClassError + \lambda \frac{N_p}{N_{max}},$$

where $\lambda = 0.01$, N_p is the number of weights the model uses, and N_{max} is the largest feasible neural network.

2.2 Possible Difficulties

While completing such a task, there are many difficulties that arise. One of these problems is the time-consuming nature of optimization; due to the fact that performing the operations necessary for evolutionary algorithms and neural networks takes a lot of time and computational power, the time constraints within this project become a barrier for determining a truly optimal solution. Furthermore, as stated in the instructions, there are 4500 possible configurations when utilizing the possible hyperparameters. This again goes back to the problem that there is not enough time to try every possible combination of hyperparameters for an adequate number of epochs. A final problem is that it is difficult to determine why the performance of the algorithm either succeeds or fails. This is a general problem in machine learning, however this makes ascribing meaning to the hyperparameters difficult.

3 Methodology

There are many types of neural networks that can be implemented, including convolutional neural networks and fully connected neural networks. The neural network that was implemented for this project was a convolutional neural network. The convolutional neural network was implemented in PyTorch, and adhered the following general architecture: a 2D convolution was applied first, followed by an activation function and pooling, wherein

it was then flattened, and a linear layer was applied, then another activation function, a final linear layer, and ultimately the log softmax function was applied.

3.1 Convolutional Neural Network

The possible hyperparameters which could be used within this architecture are stored within functions in the class "CNN_Choices". For the 2D convolutional layer, the number of filters can be either 8, 16, or 32; furthermore, the kernel can be either 3x3 with padding equal to 1 and stride equal to 1 (this means that the output will be the same size as the output (O'Shea and Nash, 2015)), or it have a kernel of size 5x5 with padding equal to 2, and stride equal to 1. The possibilities for the activation function include the ReLU, sigmoid, tanh, softplus, or ELU function. The filter size for pooling is either 2x2 or the 1x1, and pooling type is either max pooling or average pooling. The first linear layer takes as input features anywhere from 10 to 100 neurons, with each option a multiple of 10. All of these possible hyperparameter combinations result in 4500 possibilities.

3.2 Evolutionary Algorithm

In order to complete a neural architecture search, an evolutionary algorithm is employed. A crossover operator similar to that of a one-point crossover is used in the evolutionary algorithm. The crossover operator essentially randomly chooses parts of the parents and recombines them. The survival selection that is used is elitist survival selection, which copies the best performing individuals, and preserves these features. A drawback of this selection method, however, is that it can reduce the diversity of the population, and possibly lead to premature convergence (Leung and Liang, 2003). Random selection is used to select the parents as the parents are the current best from that generation. One of the mutation operators is a bit flip, which is used for binary values where a 0 is changed to a 1, and vice versa with some probability p (Gottlieb et al., 2002). The mutation operator used for the mutation of the activation function is randomly generating an integer.

4 Experiments

4.1 Description

In addition to the architecture that has been extensively discussed, two other networks have been created that were trained and tested on the same Scikit-Learn Digits dataset. These networks include a fully connected neural network, as well as a convolutional neural network. The fully connected neural network architecture of a linear layer with an input of 64 and output of 256 neurons, a ReLU activation function, another linear layer with an input size of 256 and an output of 256 neurons, another ReLU activation function, a final linear layer with an input size of 256 and an output of 10 neurons/labels, and a final log softmax function was applied to the vector. As for the convolutional neural network, the input is first reshaped, and then a 2D convolution with a kernel size of 3, stride of 1, and padding of 1 is applied, with a ReLU activation function and a max pooling layer with a kernel size of 2 and a stride of 2 applied thereafter; this convolutional layer, activation function, and pooling function is repeated once more. After this, the tensor is flattened, and then a linear layer with an input size of 256 and output size of 64 is applied, a ReLU activation function is added, another linear layer with an input of 64 neurons, and an output of 10 neurons/labels is added, and then finally the log softmax function was applied.

In order to try to minimize the impact of confounding variables, the hyperparameters of the convolutional neural network with neural architecture search kept the same hyperparameters of the other models. As such, the batch size, learning rate, and weight decay for all three are the same, at 64, 10-3, and 10-5 respectively. As for the number of epochs, 10 was decided on in order to allow for training that did not take too long, and also produced decent results. Additionally, according to (Tuite et al., 2011), training with a larger number of epochs can lead to overfitting, however, this was not a real concern.

4.2 Goals

In order to test the performance of the algorithm which was optimized using the evolutionary algorithm, the validation loss and the validation classification error are used. The classification error demonstrates the probability at which the algorithm will misclassify an instance of the data (Osisanwo et al., 2017).

The goal of this experiment is to determine whether there is a difference between the performance of a convolutional neural network that has its architecture selected by an evolutionary algorithm compared to the simpler approaches of both a fully connected neural network, and a predetermined convolutional neural network.

```
Generation: 0, best fitness: 0.03142857142857143
Generation: 2, best fitness: 0.029714285714285714
Generation: 4, best fitness: 0.029714285714285714
Generation: 6, best fitness: 0.029714285714285714
Generation: 8, best fitness: 0.029714285714285714
Generation: 10, best fitness: 0.029714285714285714
Generation: 12, best fitness: 0.029714285714285714
Generation: 14, best fitness: 0.029714285714285714
Generation: 16, best fitness: 0.029714285714285714
Generation: 18, best fitness: 0.029714285714285714
Generation: 20, best fitness: 0.029714285714285714
Generation: 22, best fitness: 0.029714285714285714
Generation: 24, best fitness: 0.029714285714285714
-> FINAL PERFORMANCE: 0.08277404921700222
```

Figure 1: Performance of algorithm

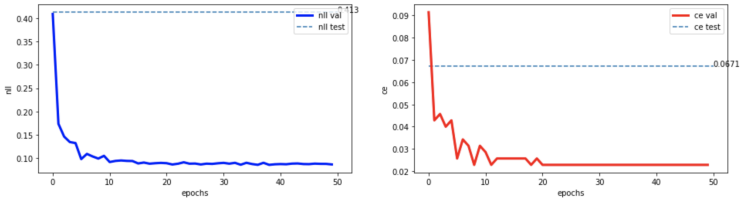

Figure 2: Graph of the performance of the algorithm

5 Results and discussion

5.1 Results

When training and testing, the resulting model appears to perform quite well. This is seen in Figure 1, where the starting fitness is already at 0.031, and by the twenty-fifth generation, the best fitness is at 0.029. While the difference between the two values is not particularly large, the starting fitness is already quite difficult to improve from.

When comparing this model (Figure 2) with the results of the fully connected neural network (Figure 3) and the convolutional neural network (Figure 4), there are some differences which can be seen. The performance of this model is worse when looking at the final performance. While the fully connected neural network had a classification error for the test set around 0.07 and the convolutional neural network had a classification error for the test set around 0.04, the current model had a classification error for the test set closer to 0.09. One explanation for this could be the fact that there were fewer generations for the model that is being tested, at 25 generations rather than 100. Another explanation could be the fact that to determine the parameters, only 10 epochs were used. Another difference between the algorithms was that the convolutional neural network with neural architecture was able to find a better model much quicker, however, had much less improvement when compared to the other models. Due to the fact that the parameters are being determined by the evolutionary algorithm, this is not that surprising.

The scatter plots in Figure 5 show what parameters were chosen by the evolutionary algorithm for the neural architecture search. The dots in red show the performance of the final generation, and by looking at these dots, there are a couple of notable observations that can be made. The first is that it is noticeable that the number of filters which was chosen most often was 32. This was not necessarily surprising as a higher filter allows for more features (Saiharsha et al., 2020), however it was interesting to see this confirmed in this data. Furthermore, it can be seen that there were three activation functions which outperformed the others, namely the ReLU, tanh, and ELU. The final two results that can be seen are that kernel shape = 3x3, stride = 1, and padding = 1, and max pooling outperformed their counterparts. The smaller kernel could be accounted for by the small dataset. Max pooling only retains the maximum values, which could indicate that the images in this dataset were not particularly complex or did not need complexity (Nagi et al., 2011), which considering the fact that the dataset is black and white and only has digits, this could make sense.

5.2 Critique & Further Work

Within this experiment, there are many deficiencies. One of these deficiencies is the lack of control within the independent variables. Comparing the performance of the three algorithms is very difficult due to the fact that there are many factors that could be contributing to the differences that are observed. An example

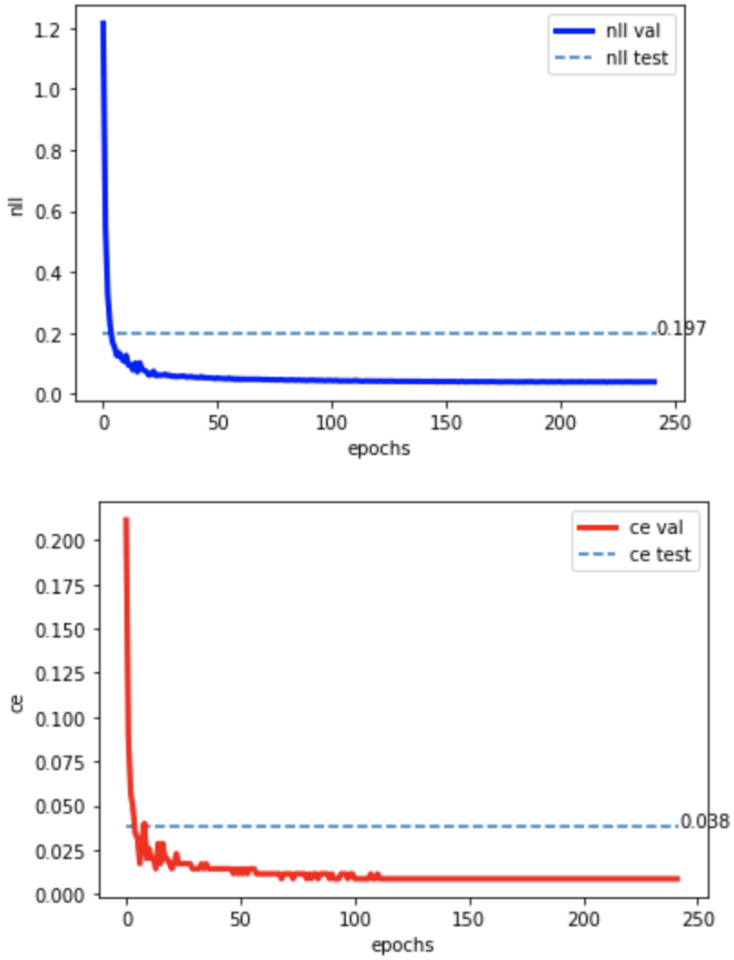

Figure 3: Graph of the fully connected neural network

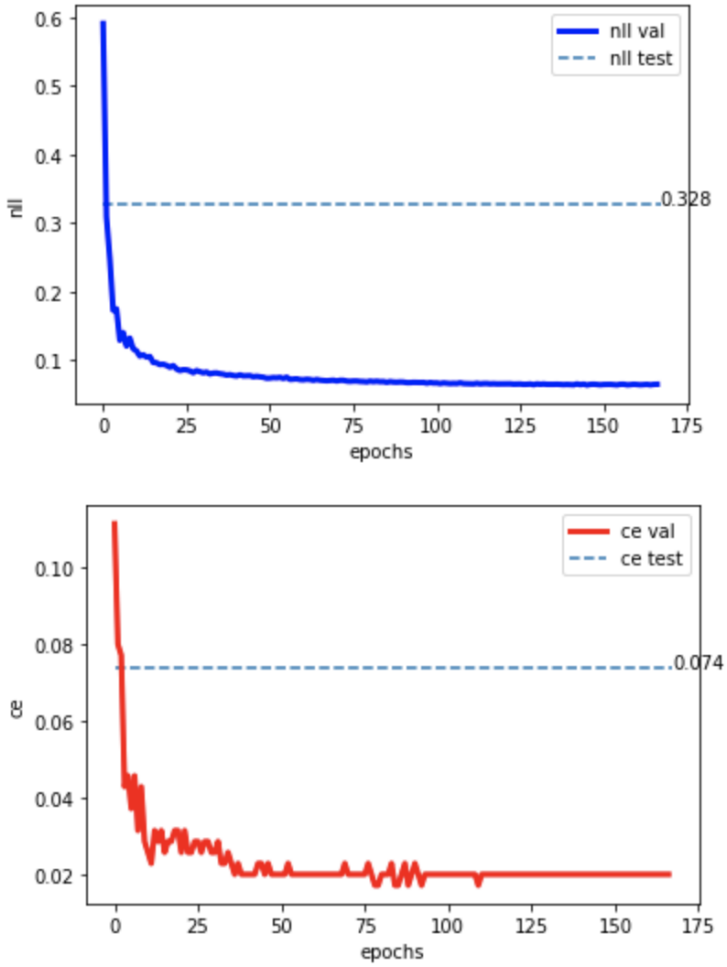

Figure 4: Graph of the convolutional neural network

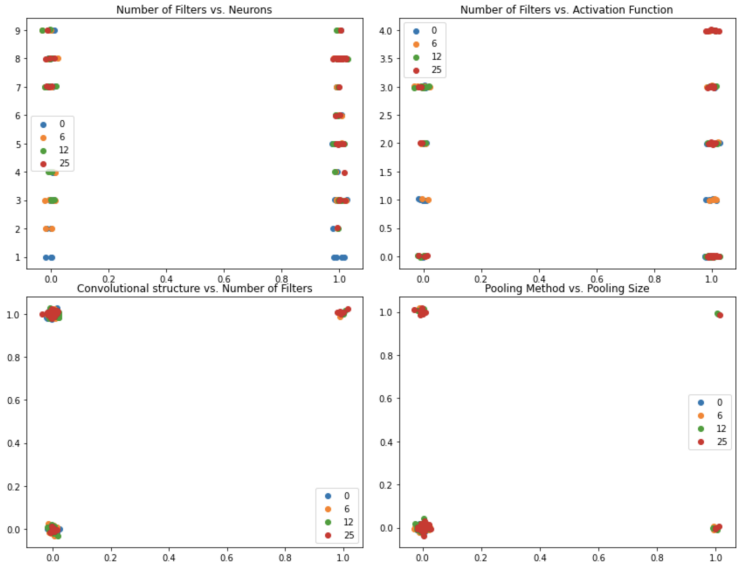

Figure 5: Scatter plots of hyperparameters

of this, is the fact that the architecture of a fully connected neural network is very different from that of a convolutional neural network. Even within the two convolutional neural networks that are studied, one has the 2D convolutional layer, activation function, and pooling repeated twice, while the convolutional neural network with neural architecture search only has this once. Within the hyperparameters, due to the fact that neural architecture search finds the optimal architecture, the architecture can be one of 4500 possibilities. A possible improvement for this experiment would be to standardize the architecture of the convolutional neural networks. A possible extension for this experiment would be to run the candidate networks for a greater number of epochs; this was not possible currently due to time constraints. Despite all of the shortcomings, a model that performed nearly as well as the comparison models was able to be found using neural architecture search.

References

Gottlieb, J., Marchiori, E., and Rossi, C. (2002). Evolutionary algorithms for the satisfiability problem. *Evolutionary computation*, 10(1):35–50.

Leung, K.-S. and Liang, Y. (2003). Adaptive elitist-population based genetic algorithm for multimodal function optimization. In *Genetic and Evolutionary Computation Conference*, pages 1160–1171. Springer.

Nagi, J., Ducatelle, F., Di Caro, G. A., Cireşan, D., Meier, U., Giusti, A., Nagi, F., Schmidhuber, J., and Gambardella, L. M. (2011). Max-pooling convolutional neural networks for vision-based hand gesture recognition. In *2011 IEEE international conference on signal and image processing applications (ICSIPA)*, pages 342–347. IEEE.

O'Shea, K. and Nash, R. (2015). An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.

Osisanwo, F., Akinsola, J., Awodele, O., Hinmikaiye, J., Olakanmi, O., and Akinjobi, J. (2017). Supervised machine learning algorithms: classification and comparison. *International Journal of Computer Trends and Technology (IJCTT)*, 48(3):128–138.

Saiharsha, B., Diwakar, B., Karthika, R., Ganesan, M., et al. (2020). Evaluating performance of deep learning architectures for image classification. In *2020 5th International Conference on Communication and Electronics Systems (ICCES)*, pages 917–922. IEEE.

Tuite, C., Agapitos, A., O'Neill, M., and Brabazon, A. (2011). Early stopping criteria to counteract overfitting in genetic programming. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, pages 203–204.