Alize Sevgi Yalçınkaya
34790
University of Michigan

March 31, 2024

ASSIGNMENT 1 REPORT — Convolutional Neural Networks

1 Part I

The objective of this project is to examine various architectures and methods for CNN image classification on the CIFAR10 dataset. The preliminary phase includes hyperparameter optimization for a simple CNN using the tensorflow library.

1.1 Preprocessing and Architecture

The dataset is preprocessed by executing three significant steps:

1. Pixel values are normalized to range between 0 and 1.
2. Labels are one-hot encoded.
3. A validation set is partitioned from the training dataset.

The model architecture incorporates two *Convolution layers* to facilitate learning of feature hierarchies. Two *MaxPooling layers* are applied post each Convolution layer to reduce feature map dimensions, thus curtailing parameter count and enhancing generalization. A *dropout layer* is integrated to combat overfitting, followed by two *dense layers* — the initial serving as a fully connected layer for learning non-linear feature combinations and the latter as the output layer with a softmax activation function.

1.2 Hyperparameter Tuning

1st Model: Basic Tuning

n_filters	kernel_size	dropout_rate
32	3	0.5
64	5	0.25

Table 1: Hyperparameter space for Model 1

Utilizing the computational resources of Google Colab's GPU, the search duration is approximately 5 minutes. Subsequent test outcomes demonstrate:

- Test loss: 0.84
- Test accuracy: 0.70

Optimal parameters identified are:

- Number of filters in the first Conv2D layer: 64
- Kernel size in Conv2D layers: 3
- Dropout rate: 0.25

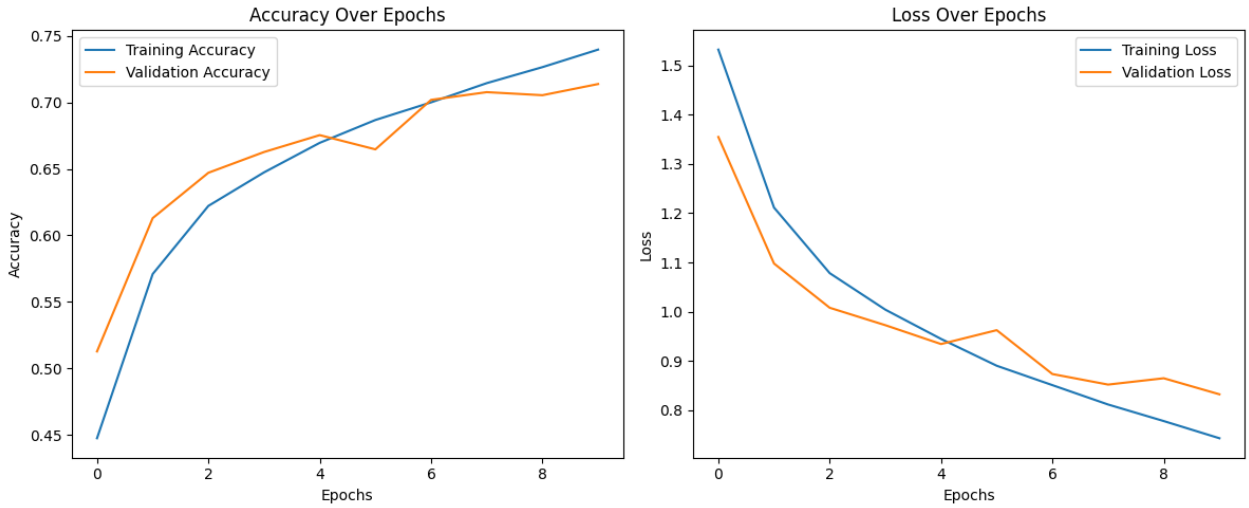

Figure 1: Accuracy and Loss for Training and Validation sets over Epochs on Model 1

The figure illustrates consistent performance improvements over epochs, with no evident signs of overfitting or underfitting.

2nd Model: Further Tuning

Informed by insights from the first model, the second model is developed to further refine complex hyperparameters.

weight_initializer	**l2_value**	**initial_learning_rate**	**learning_rate_scheduler**	**optimizer**
he_normal	min: 1e-4 (log)	min: 1e-4 (log)	constant	adam
glorot_uniform	max: 1e-2 (log)	max: 1e-2 (log)	exponential_decay	sgd
random_normal			step_decay	

Table 2: Hyperparameter space for Model 2

- Test loss: 0.83
- Test accuracy: 0.74

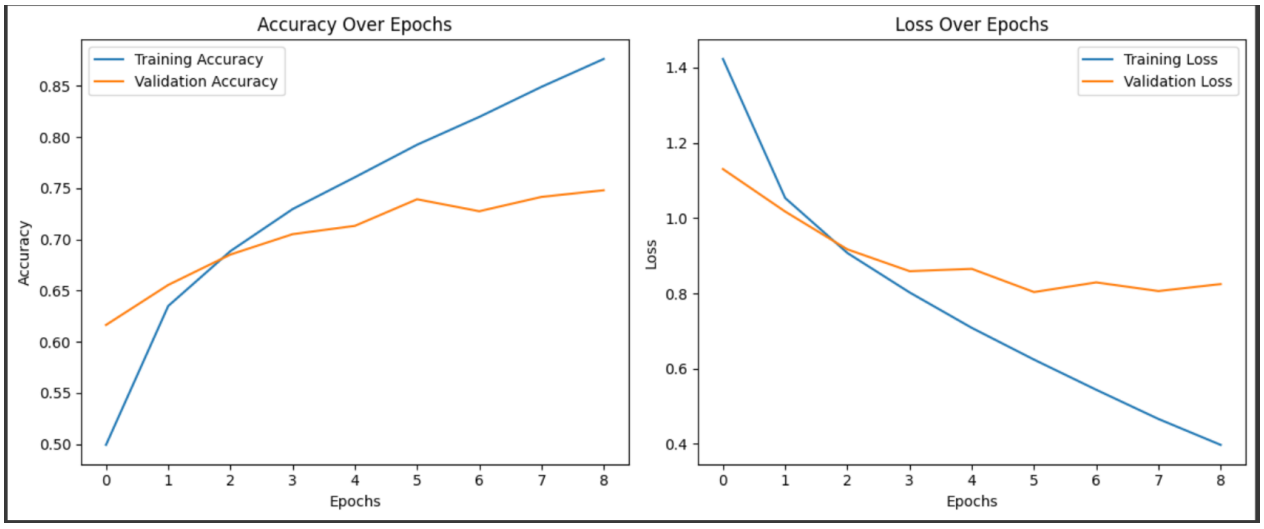

Figure 2: Accuracy and Loss for Training and Validation sets over Epochs on Model 2

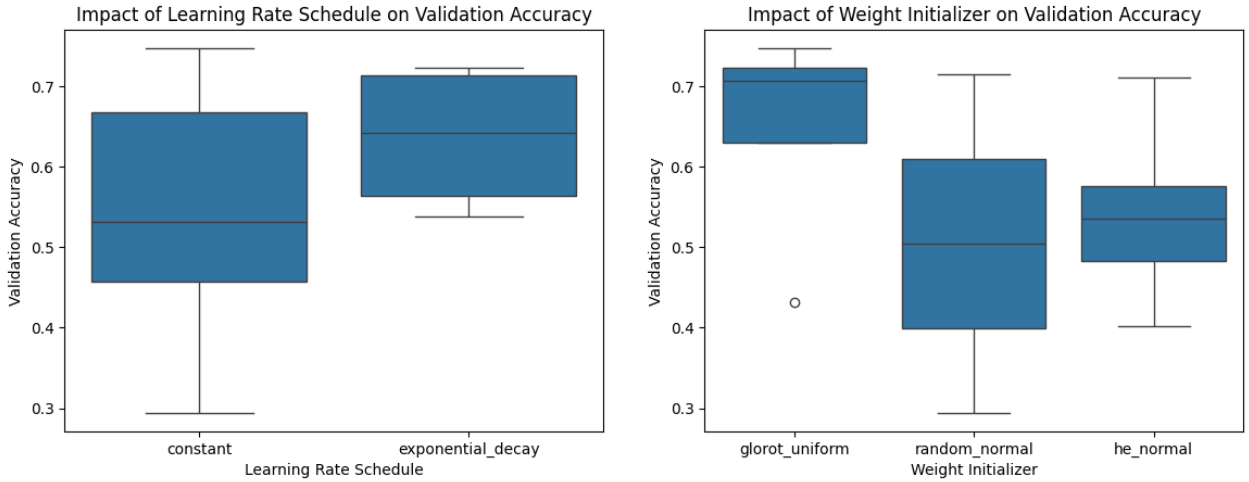

Figure 3: Effects of Learning Rate and Weight Initialization on Validation Accuracy

At times, the validation loss begins to increase slightly after the 6th epoch while the training loss continues to decrease, indicating the onset of overfitting but this does not persist across multiple executions of the code. All hyperparameters appear to contribute effectively to the model's performance. The L2 regularization value of about 0.0016 helps in mitigating overfitting but could be fine-tuned further if overfitting becomes more prominent in later epochs. If the task was to tune the model to achieve the highest accuracy deeper architectures, advanced data augmentation, or more sophisticated regularization techniques would be needed. Learning rate with exponential decay seems to yield a tighter distribution of validation accuracies around a higher mean compared to the constant decay schedule but constant decay seems to perform better in generalization and is the hyperparameter used in the best model which means it performed better on the test data but these results differed across multiple executions and exponential decay is chosen manually in light of this experience.

2 Part II

The second part of the project consists of using genetic algorithms to update the weights in our previously established CNN model. Unlike gradient-based optimization methods that adjust weights based on the gradient of a loss function, GAs search the weight space through a process of simulated evolution. To do this, custom classes were utilized for both the creation of the network and the genetic algorithm. The optimizer chosen during the hyperparameter tuning stage (ADAM) was utilized for the initialization of weights but not updates. Other hyperparameters or the overall architecture of the model were not altered.

Key Components of the Genetic Algorithm

1. **Population Initialization:** It starts with a population of randomly initialized CNN models. Each individual in the population represents a solution to the classification problem, characterized by a unique set of weights.

2. **Fitness Evaluation:** Classification accuracy on a validation dataset was chosen in line with the models from Part I.

3. **Selection:** A unique selection algorithm is employed. The accuracies (fitness) of all the models are normalized which creates a probability distribution. This means models with higher accuracies (higher fitness) have a greater chance of being selected as parents, but models with lower accuracies still have a chance, ensuring genetic diversity. Pairs of models are selected as parents based on the normalized accuracies. The indices of parent models are selected randomly and the probability of selection for each model is proportional to its normalized accuracy. This stochastic approach allows for the selection of high-performing models while maintaining some level of genetic diversity by not excluding lower-performing models entirely. [1] It was important to include less fit individuals with a non-zero probability to ensure exploration of the genetic algorithm. Without it, the algorithm couldn't have avoided premature convergence on suboptimal solutions.

```
1
2  def reproduction(self):
3      self.children_population_weights = []
4      population_idx = list(range(len(self.population)))
5      for _ in range(len(self.population)):
6          parents = np.random.choice(population_idx, size=2, replace=False,
7                                                              p=self.norm_acc)
8          parent1_weights = self.population[parents[0]].give_weights()
9          parent2_weights = self.population[parents[1]].give_weights()
10
11         mid_point = np.random.randint(0, len(parent1_weights))
12         child_weights = parent1_weights[:mid_point] + parent2_weights[mid_point:]
13         self.children_population_weights.append(child_weights)
14
15     for i, member in enumerate(self.population):
16         member.load_layer_weights(self.children_population_weights[i %
17                                   len(self.children_population_weights)])
18
```

4. **Reproduction:**A crossover point is randomly chosen. This point determines where the split between the genetic material (weights) of the two parents occurs. The weights from the first parent up to the crossover point are combined with the weights from the second parent from the crossover point onwards to form the offspring's weights. The code iterates through the population and assigns each model the weights, effectively "birthing" a new generation of models that inherit characteristics from the previous generation's most fit individuals.

5. **Mutation:**The offspring's weights are mutated with a certain probability by altering the weights of the new models.

There are several shortcomings of the model and most of these are present due to personal computational limitations. The mutation rate, population size and number of generations were not tuned but common values were used.

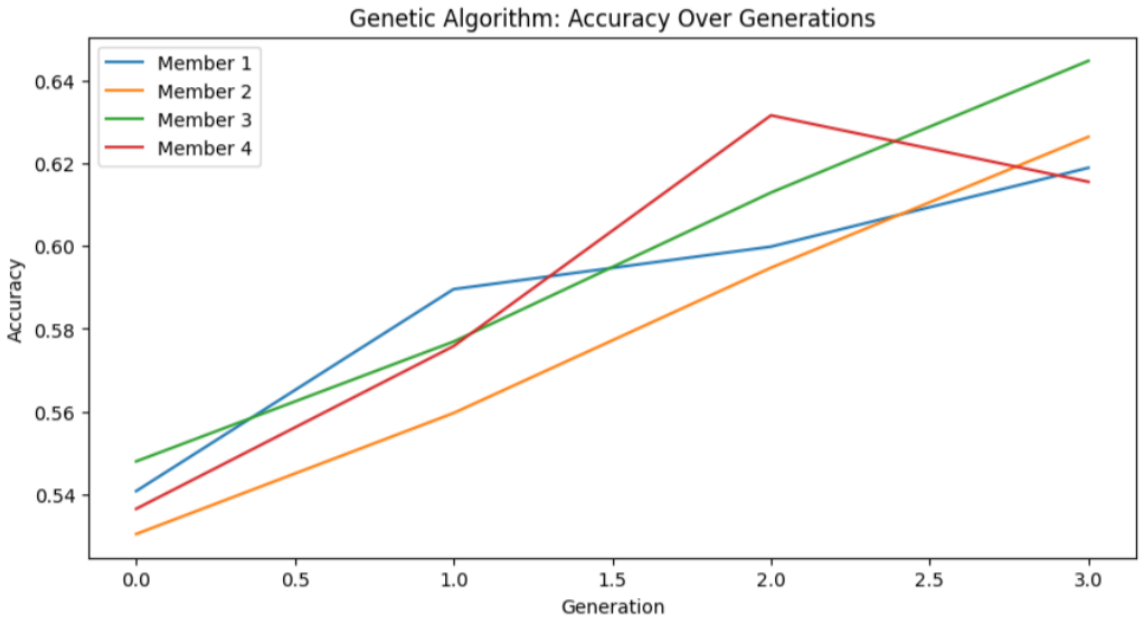

Figure 4: The evolutionary progress of the CNN models based on improvements in accuracy

The plot shows an overall increase in validation accuracy for all members across generations which indicates that the GA is successfully evolving the population correctly. There is a divergence in the accuracy trajectories of the different members suggests that the GA is maintaining a healthy diversity within the population which is crucial for avoiding local minima. Since the validation accuracy keeps increasing, it indicates that the GA is not overfitting, and the models are generalizing better as the generations proceed. The fact that accuracy continues to improve suggests that the evolutionary process has not yet converged, and running more generations might lead to further improvements.

- Test loss: 1.17
- Test accuracy: 0.65

2.1 Gradient Descent vs Genetic Algorithm

Considerations: The standard CNN is trained for 10 epochs, potentially providing more opportunity to refine its weights compared to the GA's CNNs. The GA has its own set of hyperparameters (like mutation rate and population size), which might need further tuning and that was not performed unlike for the standard CNN.

With consideration of these aspects, it is evident that the standard CNN performs better than CNN with a genetic algorithm as expected. Genetic algorithms are usually used for hyperparameter tuning rather than direct weight optimization due to the high dimensionality.

References

[1] Jonathan C.T. Kuo. Genetic algorithm in artificial neural network and its optimization methods.

[2] Nishant Manral. A guide to commonly used deep learning kernelinitializers, n.d. Retrieved March 31, 2024.

[3] Katherine (Yi) Li. How to choose a learning rate scheduler for neural networks.

[4] BY571. Genetic algorithm for neural network architecture and hyperparameter optimization and neural network weight optimization with genetic algorithm, n.d. Retrieved March 31, 2024.

Submitted by Alize Sevgi Yalçınkaya on March 31, 2024.