Artificial Intelligence coursework 2

Salva Karimisahari 240753803

Part 1:

1.1:

Optimal neural network architecture: `[{'num_neurons': 32, 'activation': 'relu', 'dropout_rate': None}]`
Best accuracy: 0.4178
Running best.py resulted in : 0.4178

```
Loaded the best model from 'best_net.pth'
Evaluation on validation set complete. Accuracy: 0.4178 (150/359 correct)
```

1.2

Encoding: The architecture of a neural network or genome is represented with a dictionary with these keys:

- num_neurons: Number of neurons in the layer.
- activation: Activation function for the layer (relu, tanh, sigmoid).
- dropout_rate: Dropout rate for the layer (can be None or a value).

For example, this is what and initial population with size 3 looks like:

```
[[{'num_neurons': 4, 'activation': 'tanh', 'dropout_rate': None}],
 [{'num_neurons': 16, 'activation': 'sigmoid', 'dropout_rate': 0.5}],
 [{'num_neurons': 8, 'activation': 'tanh', 'dropout_rate': 0.2},
  {'num_neurons': 16, 'activation': 'tanh', 'dropout_rate': 0.5},
  {'num_neurons': 16, 'activation': 'sigmoid', 'dropout_rate': None}]]
```

Selection: In this part, the given population is sorted based on the fitness scores which are also given as an input (the fitness is calculated based on the accuracy of the validation set with a selected architecture) and as much as the number of parents, the top performing individuals are returned (selected) for crossover and mutation and creating the next generation. Therefore we make sure that in each generation the best performing architectures are selected.

Crossover: The final method used to obtain the best accuracy is a combination of uniform crossover and arithmetic crossover. For each layer, the child genome randomly selects layer configurations from either parent. A random number between 0 and 1 is produced and if the random number is less than 0.5, the activation function will be the same as parent 1 and if it is more than 0.5, the activation function will be the same as parent 2.
For numerical parameters (num_neurons, dropout_rate), an arithmetic average is taken to combine the values. If a parent's dropout rate is None, the child takes the other parent's dropout rate and if they're both None, the child's dropout rate will also be None. This method

gives us higher diversity as the nature of the problem will find more accurate results if more possible architectures are explored.
One point crossover was also tested where a if both parents have more than one layer, a random crossover point is chosen and the child gets features up to the crossover point from parent 1 and the rest from parent 2. If either parent has only one layer, one of the parents will be returned. This method resulted in lower overall accuracy, mainly because it holds the structure of architectures very stable and does not result in a lot of change. Therefore the uniform method was used in the final implementation.
Example:
Parent 1: `[{'num_neurons': 4, 'activation': 'tanh', 'dropout_rate': None}]`
Parent 2: `[{'num_neurons': 16, 'activation': 'sigmoid', 'dropout_rate': 0.5}]`
Child: `[{'num_neurons': 10, 'activation': 'tanh', 'dropout_rate': 0.5}]`

Mutation: A rate of 0.3 was defined for mutation. Meaning that if a produced random number is less than 0.3, one layer attribute is chosen to be mutated. These attributes can be the number of neurons, the activation function or the dropout. If the selected mutation type is the number of neurons, the child's number of neurons is randomly selected from [8,16,32,64]. If the selected mutation type is an activation function, the child's activation function is randomly chosen from ['relu' , 'tanh', 'sigmoid']. If the chosen mutation is dropout, the child's dropout value will be chosen from [None, 0.2, 0.5]. Additionally, to ensure exploration of architectures with different depths, with a probability of mutation rate, a layer gets added or removed to the genome. So If a random number between 0 and 1 is less than 0.3, a layer with a random number of neurons from the list [8,16,32,64], a random activation function chosen from the list ['relu' , 'tanh', 'sigmoid'] and a random dropout chosen from the list [None, 0.2, 0.5] is added to as a layer or one of the existing layers is removed (if it doesn't result in an empty architecture).
Example:
Child: `[{'num_neurons': 10, 'activation': 'tanh', 'dropout_rate': 0.5}]`
Mutated child (run 1): `[{'num_neurons': 10, 'activation': 'tanh', 'dropout_rate': 0.5}]`

Mutated child (run 5): `[{'num_neurons': 10, 'activation': 'sigmoid', 'dropout_rate': 0.5}]`

1.3

Many different combinations of population size, number of generations, number of parents and architecture were tested. Some of the sample blueprints and their results are shown below:

```
blueprint = {
 'max_n_layers': 4,           # Define the (maximum) number of layers in each
neural network
   'neurons': [4,8,16],              # Possible neuron counts per layer
   'activations': ['sigmoid',"relu","tanh"],  # Possible activation functions
   'dropout': [None,0.2,0.5]            # Possible dropout rates, including 'None'
for no dropout
```

```
}
```

```
Final Summary
Best Overall Fitness: 0.362116991643454
Best Overall Architecture: [{'num_neurons': 8, 'activation': 'relu',
'dropout_rate': None}]

Starting the re-training of the best model found by the genetic algorithm
(corroborate reproducibility)
Epoch 1/5 complete. Average Training Loss: 1.7369
Epoch 2/5 complete. Average Training Loss: 1.6302
Epoch 3/5 complete. Average Training Loss: 1.5712
Epoch 4/5 complete. Average Training Loss: 1.5377
Epoch 5/5 complete. Average Training Loss: 1.5137
Evaluation on validation set complete. Accuracy: 0.3928 (141/359 correct)
Saved the best model's weights to 'best_net.pth'
```

Since the previous neural network architecture was overly simple and the accuracy was low, additional scenarios with added layers, increased number of layers and increased population were tested:

```
 blueprint = {
  'max_n_layers': 5,           # Define the (maximum) number of layers in each
neural network
   'neurons': [8, 16, 32, 64, 128],              # Possible neuron counts per layer
   'activations': ['sigmoid',"relu","tanh"],  # Possible activation functions
   'dropout': [None,0.1,0.2,0.3,0.5]             # Possible dropout rates, including
'None' for no dropout
}
```

```
Final Summary
Best Overall Fitness: 0.281932123215631
Best Overall Architecture: [{'num_neurons': 64, 'activation': 'relu',
'dropout_rate': 0.2}]

Starting the re-training of the best model found by the genetic algorithm
(corroborate reproducibility)
Epoch 1/5 complete. Average Training Loss: 1.6369
Epoch 2/5 complete. Average Training Loss: 1.5614
Epoch 3/5 complete. Average Training Loss: 1.6113
Epoch 4/5 complete. Average Training Loss: 1.5035
Epoch 5/5 complete. Average Training Loss: 1.5012
Evaluation on validation set complete. Accuracy: 0.3342 (120/359 correct)
Saved the best model's weights to 'best_net.pth'
```

The accuracy got worse after making these changes, suggesting that the modifications might have made the structure more complex without improving the performance.

After running tests on other architectures, it was concluded that this combination got to the best result as it had enough generation and population size to converge to an optimal result:

```
population_size = 20
num_generations = 15
num_parents = 5
```

The blueprint chosen among different tests is:

```
blueprint = {
   'max_n_layers': 5,           # Define the (maximum) number of layers in each
neural network
   'neurons': [8,16,32,64,128],              # Possible neuron counts per layer
   'activations': ['sigmoid',"relu","tanh"],  # Possible activation functions
   'dropout': [None,0.2,0.4]            # Possible dropout rates, including 'None'
for no dropout
}
```

As less than 5 layers resulted in very simple architectures that didn't capture the dataset's patterns, less than 5 neurons resulted in underfitting and more than 128 was probably overfitting, dropout values were always chosen in this range even when a more diverse list of all values between 0 and 0.5 were given. The optimal output was achieved on the first run and running the experiment 9 additional times resulted in these outputs:

Run 1 (Best accuracy):

```
Final Summary
Best Overall Fitness: 0.44846796657381616
Best Overall Architecture: [{'num_neurons': 32, 'activation': 'relu',
'dropout_rate': None}]

Starting the re-training of the best model found by the genetic algorithm
(corroborate reproducibility)
Epoch 1/5 complete. Average Training Loss: 1.8755
Epoch 2/5 complete. Average Training Loss: 1.7975
Epoch 3/5 complete. Average Training Loss: 1.7181
Epoch 4/5 complete. Average Training Loss: 1.6716
Epoch 5/5 complete. Average Training Loss: 1.6356
Evaluation on validation set complete. Accuracy: 0.4178 (150/359 correct)
Saved the best model's weights to 'best_net.pth'
```

Run 2:

```
Final Summary
Best Overall Fitness: 0.41225626740947074
Best Overall Architecture: [{'num_neurons': 60, 'activation': 'relu',
'dropout_rate': 0.25}, {'num_neurons': 56, 'activation': 'sigmoid', 'dropout_rate':
0.2}, {'num_neurons': 16, 'activation': 'sigmoid', 'dropout_rate': None}]

Starting the re-training of the best model found by the genetic algorithm
(corroborate reproducibility)
Epoch 1/5 complete. Average Training Loss: 1.7279
Epoch 2/5 complete. Average Training Loss: 1.5804
Epoch 3/5 complete. Average Training Loss: 1.5065
Epoch 4/5 complete. Average Training Loss: 1.4707
Epoch 5/5 complete. Average Training Loss: 1.4514
Evaluation on validation set complete. Accuracy: 0.3928 (141/359 correct)
Saved the best model's weights to 'best_net_normal.pth'
```

Run 3:

```
Final Summary
Best Overall Fitness: 0.403899721448468
Best Overall Architecture: [{'num_neurons': 49, 'activation': 'relu',
'dropout_rate': None}]
```

```
Starting the re-training of the best model found by the genetic algorithm
(corroborate reproducibility)
Epoch 1/5 complete. Average Training Loss: 2.0394
Epoch 2/5 complete. Average Training Loss: 1.9666
Epoch 3/5 complete. Average Training Loss: 1.8837
Epoch 4/5 complete. Average Training Loss: 1.8565
Epoch 5/5 complete. Average Training Loss: 1.7870
Evaluation on validation set complete. Accuracy: 0.4039 (145/359 correct)
Saved the best model's weights to 'best_net_normal.pth'
```

Run 4:

```
Final Summary
Best Overall Fitness: 0.42618384401114207
Best Overall Architecture: [{'num_neurons': 42, 'activation': 'relu',
'dropout_rate': 0.2}]

Starting the re-training of the best model found by the genetic algorithm
(corroborate reproducibility)
Epoch 1/5 complete. Average Training Loss: 2.0387
Epoch 2/5 complete. Average Training Loss: 2.0387
Epoch 3/5 complete. Average Training Loss: 1.9170
Epoch 4/5 complete. Average Training Loss: 1.9230
Epoch 5/5 complete. Average Training Loss: 1.9003
Evaluation on validation set complete. Accuracy: 0.3370 (121/359 correct)
Saved the best model's weights to 'best_net_normal.pth'
```

Run 5:

```
Final Summary
Best Overall Fitness: 0.3983286908077994
Best Overall Architecture: [{'num_neurons': 64, 'activation': 'relu',
'dropout_rate': 0.2}]

Starting the re-training of the best model found by the genetic algorithm
(corroborate reproducibility)
Epoch 1/5 complete. Average Training Loss: 2.3602
Epoch 2/5 complete. Average Training Loss: 2.3877
Epoch 3/5 complete. Average Training Loss: 2.2372
Epoch 4/5 complete. Average Training Loss: 2.2110
Epoch 5/5 complete. Average Training Loss: 2.1647
Evaluation on validation set complete. Accuracy: 0.3788 (136/359 correct)
Saved the best model's weights to 'best_net_normal.pth'
```

Run 6:

```
Final Summary
Best Overall Fitness: 0.43454038997214484
Best Overall Architecture: [{'num_neurons': 28, 'activation': 'relu',
'dropout_rate': None}]

Starting the re-training of the best model found by the genetic algorithm
(corroborate reproducibility)
Epoch 1/5 complete. Average Training Loss: 1.8554
Epoch 2/5 complete. Average Training Loss: 1.7433
Epoch 3/5 complete. Average Training Loss: 1.6922
Epoch 4/5 complete. Average Training Loss: 1.6249
Epoch 5/5 complete. Average Training Loss: 1.6128
Evaluation on validation set complete. Accuracy: 0.3788 (136/359 correct)
Saved the best model's weights to 'best_net_normal.pth'
```

Run 7:

```
Final Summary
Best Overall Fitness: 0.41225626740947074
```

```
Best Overall Architecture: [{'num_neurons': 36, 'activation': 'relu',
'dropout_rate': 0.25}]

Starting the re-training of the best model found by the genetic algorithm
(corroborate reproducibility)
Epoch 1/5 complete. Average Training Loss: 2.0177
Epoch 2/5 complete. Average Training Loss: 1.9362
Epoch 3/5 complete. Average Training Loss: 1.9214
Epoch 4/5 complete. Average Training Loss: 1.8837
Epoch 5/5 complete. Average Training Loss: 1.8412
Evaluation on validation set complete. Accuracy: 0.3844 (138/359 correct)
Saved the best model's weights to 'best_net_normal.pth'
```

Run 8:

```
Final Summary
Best Overall Fitness: 0.43454038997214484
Best Overall Architecture: [{'num_neurons': 28, 'activation': 'relu',
'dropout_rate': None}]

Starting the re-training of the best model found by the genetic algorithm
(corroborate reproducibility)
Epoch 1/5 complete. Average Training Loss: 1.8397
Epoch 2/5 complete. Average Training Loss: 1.7188
Epoch 3/5 complete. Average Training Loss: 1.6669
Epoch 4/5 complete. Average Training Loss: 1.6085
Epoch 5/5 complete. Average Training Loss: 1.5771
Evaluation on validation set complete. Accuracy: 0.4067 (146/359 correct)
Saved the best model's weights to 'best_net_normal.pth'
```

Run 9:

```
Final Summary
Best Overall Fitness: 0.44846796657381616
Best Overall Architecture: [{'num_neurons': 39, 'activation': 'relu',
'dropout_rate': None}]

Starting the re-training of the best model found by the genetic algorithm
(corroborate reproducibility)
Epoch 1/5 complete. Average Training Loss: 1.9344
Epoch 2/5 complete. Average Training Loss: 1.8323
Epoch 3/5 complete. Average Training Loss: 1.7970
Epoch 4/5 complete. Average Training Loss: 1.7376
Epoch 5/5 complete. Average Training Loss: 1.7275
Evaluation on validation set complete. Accuracy: 0.3928 (141/359 correct)
Saved the best model's weights to 'best_net_normal.pth'
```

Run 10:

```
Final Summary
Best Overall Fitness: 0.4596100278551532
Best Overall Architecture: [{'num_neurons': 80, 'activation': 'relu',
'dropout_rate': None}]

Starting the re-training of the best model found by the genetic algorithm
(corroborate reproducibility)
Epoch 1/5 complete. Average Training Loss: 2.3988
Epoch 2/5 complete. Average Training Loss: 2.4047
Epoch 3/5 complete. Average Training Loss: 2.3417
Epoch 4/5 complete. Average Training Loss: 2.1825
Epoch 5/5 complete. Average Training Loss: 2.1268
Evaluation on validation set complete. Accuracy: 0.3983 (143/359 correct)
Saved the best model's weights to 'best_net_normal.pth'
```

Overall, the fitness values range from 0.3983 to 0.4596 in different runs with the same blueprint and hyperparameters. Runs 6 and 8 ({'num_neurons': 28, 'activation': 'relu', 'dropout_rate': None})were the only ones that appeared twice. Although similar architectures are observed since the exact same architecture occurred only twice in 10 tims, we can report the reproduction rate as 10%.
Run 6 ({'num_neurons': 28, 'activation': 'relu', 'dropout_rate': None})and run 9 (9:{'num_neurons': 39, 'activation': 'relu', 'dropout_rate': None}) were the most similar architectures to our best model ({'num_neurons': 32, 'activation': 'relu', 'dropout_rate': None}) and had daily high accuracies as well. In general, obtained architectures have slight differences but the 'relu' activation function, no dropouts (the dataset is probably not big enough for scenarios of overfitting)and 28-64 neurons (so moderate complexity to capture dataset's underlying patterns) are seen quite frequently, meaning there is a pattern in the optimization process and the genetic algorithms consistently explores a range of similar effective solutions. As expected, the genetic algorithm doesn't alway converge to the same architecture because of the randomness in the crossover and mutation process. We can also see that increased number of neurons (run 2: 60 neurons and run 10: 80 neurons) doesn't add to the accuracy and fitness score while increasing the complexity much more. The architecture obtained in the 10th run with 80 neurons achieves the highest fitness score and it has common elements (relu and no dropout) with the architecture with highest accuracy.

1.4

The same parameters used in the previous question with uniform crossover and the accuracy dropped significantly:

```
Final Summary
Best Overall Fitness: 0.3983286908077994
Best Overall Architecture: [{'num_neurons': 64, 'activation': 'relu',
'dropout_rate': None}]

Starting the re-training of the best model found by the genetic algorithm
(corroborate reproducibility)
Epoch 1/5 complete. Average Training Loss: 2.2440
Epoch 2/5 complete. Average Training Loss: 2.1541
Epoch 3/5 complete. Average Training Loss: 2.0918
Epoch 4/5 complete. Average Training Loss: 2.0627
Epoch 5/5 complete. Average Training Loss: 1.9838
Evaluation on validation set complete. Accuracy: 0.3621 (130/359 correct)
Saved the best model's weights to 'best_net_normal.pth'
```

We will explore the effect of different population sizes with out best architecture which looked like this:

```
num_generations = 15
num_parents = 5
blueprint = {
'max_n_layers': 5,           # Define the (maximum) number of layers in each
neural network
  'neurons': [8,16,32,64,128],              # Possible neuron counts per layer
  'activations': ['sigmoid',"relu","tanh"],  # Possible activation functions
```

```
    'dropout': [None,0.2,0.4]           # Possible dropout rates, including 'None'
for no dropout
}
```

Population size	Accuracy
5	0.3278
10	0.2730
15	0.3844
20	0.4178

Population size is the parameter that affects the diversity of the solutions explored. Running the algorithm multiple times with different population sizes of [5, 10, 15, 20] with everything else being fixed can show us these observations:
The smaller population sizes like 5 and 10 result in lower accuracy since this number limits the diversity of the genetic algorithm. So instead of exploring a population with many options, the algorithm will have to converge to a suboptimal solution. Increasing to a moderate population size (15) increases the accuracy as it allows more diverse options in the mating pool and a larger population size that got the best performing accuracy has an increased computational time but the diversity allows the algorithm to explore the search space thoroughly and find the optimal solution.

Part 2

2.1

Since we are dealing with multi-class classification, it is best to use cross-entropy loss since it measures how the predicted probability distribution matches the target distribution.
A training example x with a label y and a class scores vector of z = $(z_1,z_2,...,z_k)$ for k output classes, the predicted probability for class j is the softmax probability:

$$p_j = \frac{exp(z_j)}{\sum_{1}^{k} exp(z_i)}$$

The cross entropy loss for x will be -log(p_y) and the loss for each batch of N can be computed with:

$$L_{batch} = -\frac{1}{N}\sum_{i=1}^{N} y_i log(p_{y_i})$$

p_{y_i} is the probability of the actual label for x_i an y_i is the actual label of x_i
The gradient with respect to class scores:

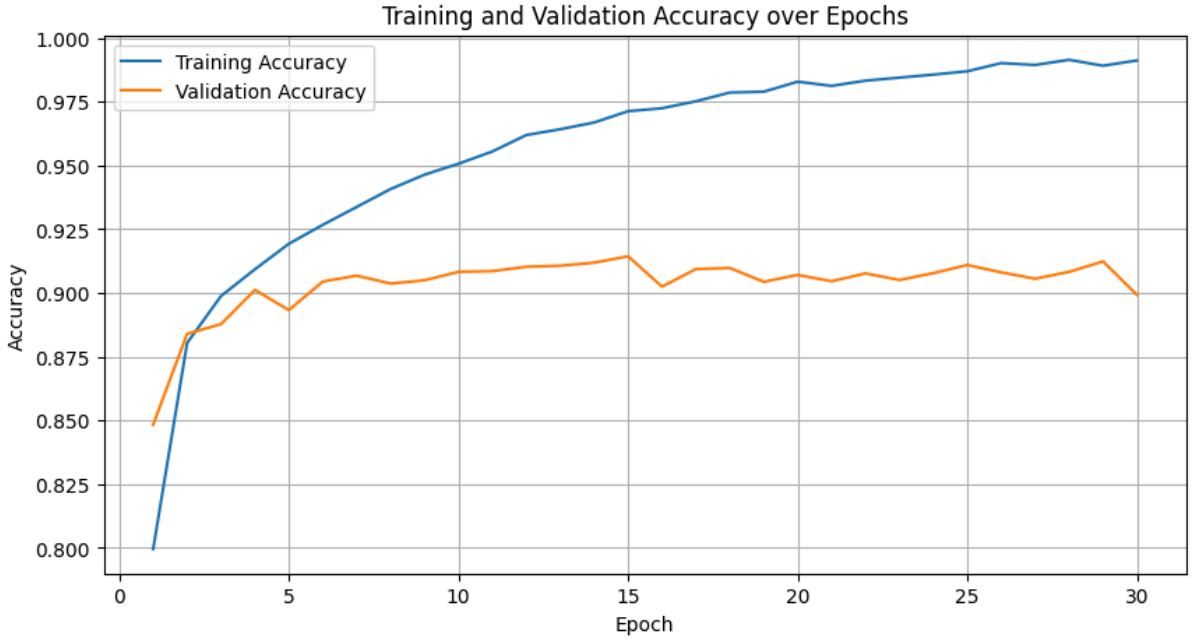
$$\frac{\partial L}{\partial z_j} = p_j - y_j$$

The gradient is the difference between the predicted probability for a class and the actual label, so if the model predicts the correct class with high probability the gradient magnitude w.r.t. That class's logit will be small meaning we are close to the solution but if the model

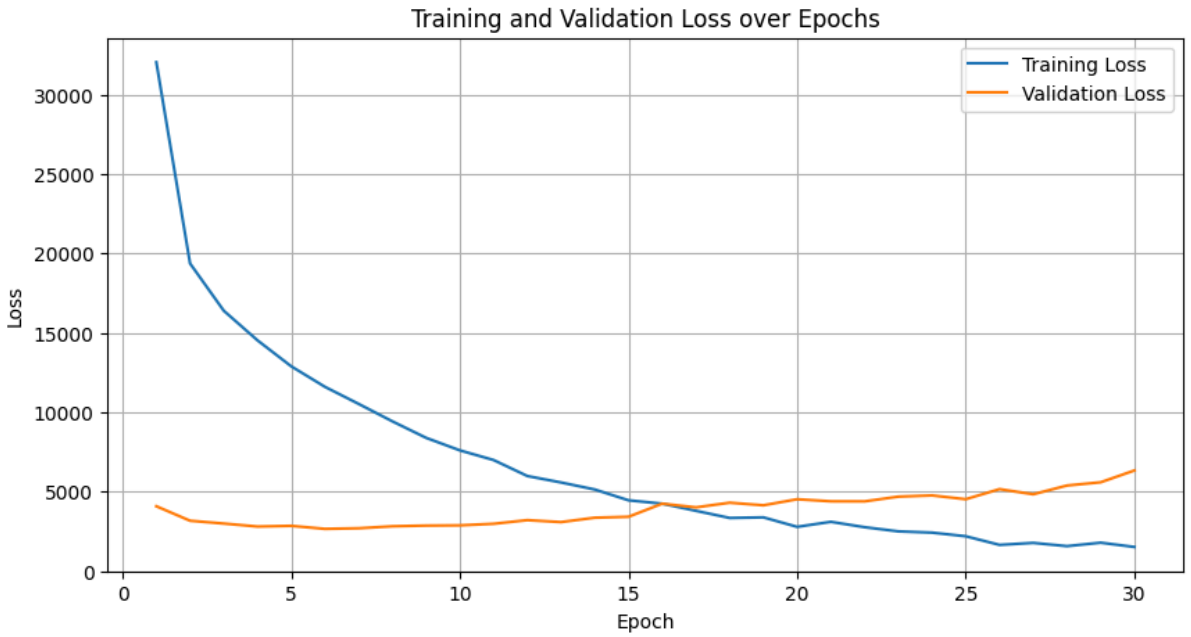
predicts the correct class with low probability, the gradient w.r.t. of the correct class logit is large and positive so it will push the model to minimize it.

2.2

A. Final train and validation accuracy
   Train and validation accuracy in the last epoch are:
   train accuracy: 0.9911833333333333
   validation accuracy: 0.8
B. Plot of the accuracy on the training and validation sets per epoch



   Training accuracy: in the first few epochs, the training accuracy increases dramatically (from 0.8 to 0.92 in just 5 epochs) and then continues increasing at a lower speed until it reaches a value close to 1 by the end of 30 epochs, which indicates the model is prone to overfit on the training data.
   Validation accuracy: The validation accuracy increases initially but starts fluctuating between 0.88 and 0.92 and never reaches the training accuracy and does not mimic its behaviour. This shows that the model does not improve its predictions for unseen data after a certain point of training and early stopping or a less comple model might be a better solution for the dataset.
C. Plot of the train and validation loss per epoch



Training loss: The training loss starts very high and experiences a rapid decrease over the first few epochs because the model is beginning to fit the data. As training progresses, the training loss is continuously decreasing but with a slower pace until it approaches its minimum by the end of 30 epochs showing that the model can make good predictions on the training set.
Validation Loss: The validation loss starts much lower than the training loss, decreases a bit initially and then flattens out. After 15 epochs, it was expected for it to decrease however it started increasing meaning that even though the model is getting better at reducing the loss on train set predictions, it is not improving on generalizing the validation set. This is a sign of our model overfitting on the training dataset because it's memorizing the patterns of the training set and therefore not generalizing well. The model should work better if it has different hyperparameters or a smaller learning rate so it doesn't memorize the trining data and just learns the underlying patterns.

Part 3

3.1

1. Take whole apple:
    - Pre-conditions:
        - The apple must be available so action "take" can be done
    - Effects:
        - The apple will be in the robot's hand (in_hand)
        - The apple will not be available

2. Peel apple:
    - Pre-conditions:
        - The apple must be clean
        - Apple must not be already peeled

        - Apple must be in hand
        - Robot must have a knife
    - Effects:
        - The apple will be peeled
3. Cut apple:
    - Pre-conditions:
        - The apple must be peeled
        - Apple must not be already cut
        - Apple must be in hand
        - Robot must have a knife
    - Effects:
        - The apple will be cut
4. Put down knife:
    - Pre-conditions:
        - Robot must have a knife in hand
    - Effects:
        - Knife won't be in robot's hand
        - Knife will be available
5. Add chopped apple to container:
    - Pre-conditions:
        - Chopped apple must be in robot's hand
        - Container must be empty
    - Effects:
        - Container will contain chopped apple
        - Container will not be empty
        - Chopped apple will not be in robot's hand

3.2

1. Take whole apple: The apple is clean, unpeeled and uncut and not in the robot's hand
    - is_clean(apple)
    - not(is_peeled(apple))
    - not(is_cut(apple))
    - available(apple)
    - available(knife)
    - available(apple)
    - available(container)
    - empty(container)
    - not(in_hand(apple))
    - not(in_hand(knife))
2. Peel apple: The robot has the apple and the knife in its hands, the apple is clean and unpeeled.
    - is_clean(apple)
    - not(available(apple))
    - in_hande(apple)
    - in_hand(knife)

- not(is_peeled(apple))
- not(is_cut(apple))
- empty(container)
- available(container)
- not(available(knife))

3. Cut apple: The peeled apple and the knife are in the robot's hands
    - is_clean(apple)
    - is_peeled(apple)
    - not(is_cut(apple))
    - in_hand(apple)
    - in_hand(knife))
    - empty(container)
    - available(container)
4. Put down knife: The apple has been cut and it is in the robot's hand. The robot is still holding the knife.
    - is_clean(apple)
    - is_peeled(apple)
    - is_cut(apple)
    - in_hand(apple)
    - in_hand(knife)
    - available(container)
    - empty(container)
5. Add chopped apple to container: The robot has put down the knife, the apple is cut.
    - is_clean(apple)
    - is_peeled(apple)
    - is_cut(apple)
    - in_hand(apple)
    - available(knife)
    - not(in_hand(knife))
    - empty(container)
    - available(container)

3.3

1. The action of peeling an apple
    - Preconditions of peeling an apple:
        - is_clean(x)
        - in_hand(x) (apple)
        - In_hand(y) (knife)
        - ¬is_peeled(x)
    - Effect of peeling an apple:
        - is_peeled(x)
    - FOL

      For all x and y, if the apple x is clean, not peeled, in hand and knife y is in hand, then x becomes peeled

      $\forall x , \forall y [(\text{is\_clean}(x) \wedge \text{in\_hand}(x) \wedge \text{in\_hand}(y) \wedge \neg\text{is\_peeled}(x) \rightarrow \text{is\_peeled}(x)]$
    - CNF

- Eliminating '→' (P → Q ≡ ¬P ∨ Q)
  ∀x , ∀y [¬(is_clean(x) ∧ in_hand(x) ∧ in_hand(y) ∧ ¬is_peeled(x)) ∨ is_peeled(x)]
- Move ¬ inwards
  ∀x , ∀y [(¬is_clean(x) ∨ ¬in_hand(x) ∨ ¬in_hand(y) ∨ is_peeled(x)) ∨ is_peeled(x)]
- Combine all ORs and simplify (final CNF)
  ∀x , ∀y [¬is_clean(x) ∨ ¬in_hand(x) ∨ ¬in_hand(y) ∨ is_peeled(x)]

2. The action of cutting an apple
   - Preconditions for cutting an apple:
     - is_clean(x)
     - in_hand(x) (apple)
     - In hand(y) (knife)
     - ¬is_cut(x)
     - is_peeled(x)
   - Effects of cutting an apple
     - is_cut(x)
   - FOL

     For all x and y, if the apple x is clean, peeled, in hand, uncut and if the knife y is in hand, then x is cut.
     ∀x , ∀y [(is_clean(x) ∧ is_peeled(x) ∧ in_hand(x) ∧ in_hand(y) ∧ ¬ is_cut(x))→ is_cut(x)]

   - CNF
     - Eliminating '→' (P → Q ≡ ¬P ∨ Q)
       ∀x , ∀y [¬(is_clean(x) ∧ is_peeled(x) ∧ in_hand(x) ∧ in_hand(y) ∧ ¬ is_cut(x)) ∨ is_cut(x)]
     - Move ¬ inwards
       ∀x , ∀y [(¬is_clean(x) ∨ ¬is_peeled(x) ∨ ¬in_hand(x) ∨ ¬ in_hand(y) ∨ is_cut(x)) ∨ is_cut(x)]
     - Combine all ORs and simplify (final CNF)
       ∀x , ∀y [¬is_clean(x) ∨ ¬is_peeled(x)∨ ¬in_hand(x) ∨ ¬in_hand(y) ∨ is_cut(x)]

3.4

1. To prepare the apple salad, we take these steps:
   a. Take a whole, unpeeled, clean and available apple in hand
   b. Hold the apple and the available knife
   c. Peel the apple with the knife
   d. Hold the apple and the knife
   e. Cut the apple with the knife
   f. Put the knife down
   g. Add chopped apples in hand to empty container

2. Our initial conditions include:

- is_clean(apple)
- ¬is_peeled(apple)
- ¬is_cut(apple)
- available(apple)
- available(knife)
- empty(container)

After having these initial conditions, next steps will happen in sequence:

A. We take the clean, un peeled and un cut apple in hand: in_hand(apple) → proposition: TAKE_APPLE
B. After having the knife and apple in hand, we can peel the apple: is_peeled(apple) → proposition: PEEL_APPLE
C. After having the peeled apple in hand and holding the knife in the other hand, we can cut the apple: is_cut(apple) → proposition: CUT_APPLE
D. We put the knife down after cutting: available(knife) → proposition: PUT_KNIFE_DOWN
E. The container is empty and we have apple slices in hand so we can add the apples: contains(container, apple), empty(container), ¬in_hand(apple) → proposition: ADD_TO_CONTAINER

3. LTL:

Initial_conditions := is_clean(apple) ∧ ¬is_peeled(apple) ∧ ¬is_cut(apple) ∧ available(apple) ∧ available(knife) ∧ empty(container)

G(initial_condition→
X(TAKE_APPLE →
X(PEEL_APPLE →
X(CUT_APPLE→
X(PUT_KNIFE_DOWN)→
X(ADD_TO_CONTAINER)
)
)
)
)
)