Cathy Wang
CS294-082

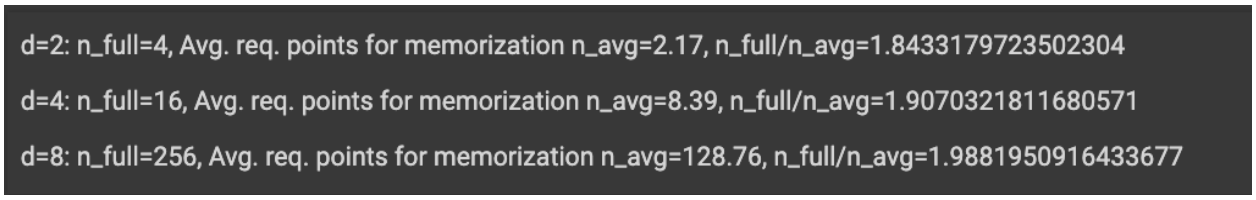
Midterm Outputs and Writeup

(Note: see jupyter notebook in repository for codes)

Chapter 6

Exercise 6.1:

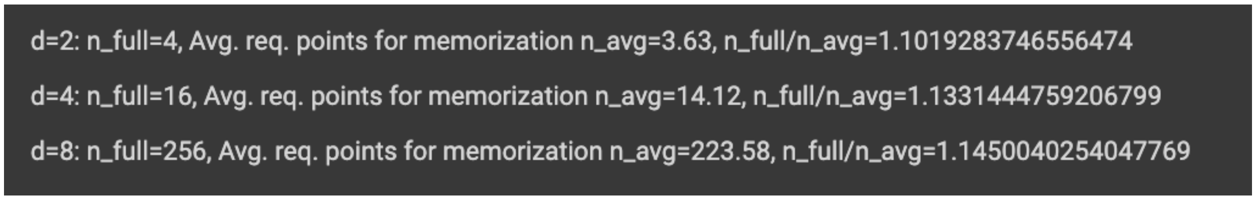
a) Output:



```
d=2: n_full=4, Avg. req. points for memorization n_avg=2.17, n_full/n_avg=1.8433179723502304
d=4: n_full=16, Avg. req. points for memorization n_avg=8.39, n_full/n_avg=1.9070321811680571
d=8: n_full=256, Avg. req. points for memorization n_avg=128.76, n_full/n_avg=1.9881950916433677
```

Since n_full/n_avg is ~ 2 and also approaches 2 as d increases, the information limit of 2 bits per parameter holds for KNN.

b) Output:



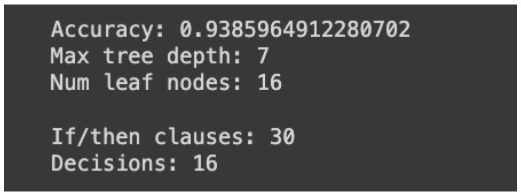
```
d=2: n_full=4, Avg. req. points for memorization n_avg=3.63, n_full/n_avg=1.1019283746556474
d=4: n_full=16, Avg. req. points for memorization n_avg=14.12, n_full/n_avg=1.1331444759206799
d=8: n_full=256, Avg. req. points for memorization n_avg=223.58, n_full/n_avg=1.1450040254047769
```

The expected n_full/n_avg is c/c-1 (compression). My result here is a simulation for 8 classes, so you would expect it to approach 1.14 as d increases. You can see this pattern happening here in my output.

Exercise 6.2:

a) For this exercise, I used sklearn’s DecisionTreeClassifier. To simplify the tree, I focused on how my strategies affect **both** the number of if/then clauses (defined as “<=” or “>” clauses that make a decision) and decisions (class label). I found the best model (first prioritizing same or better accuracy as the original, then prioritizing minimizing clauses) by testing all different possible sizes of max_depth and max_leaf_nodes.

Dataset #1: breast cancer (binary classification)
Original tree:

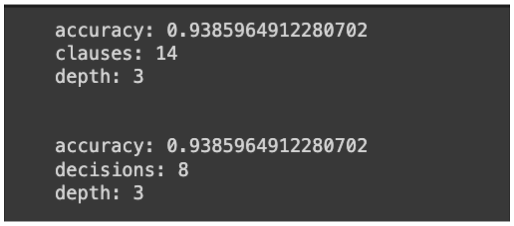


```
Accuracy: 0.9385964912280702
Max tree depth: 7
Num leaf nodes: 16

If/then clauses: 30
Decisions: 16
```

Strategies employed and best result:

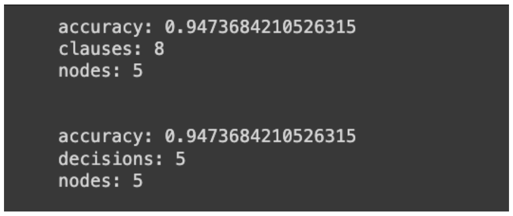
- Tune max_depth of tree



```
accuracy: 0.9385964912280702
clauses: 14
depth: 3


accuracy: 0.9385964912280702
decisions: 8
depth: 3
```

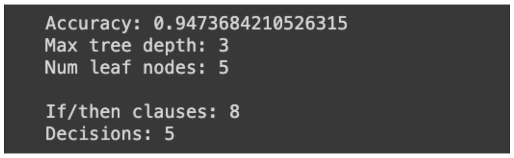
- Tune max_leaf_nodes of tree



```
accuracy: 0.9473684210526315
clauses: 8
nodes: 5


accuracy: 0.9473684210526315
decisions: 5
nodes: 5
```

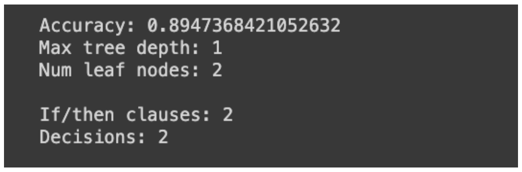
- Best result overall (prioritizing both accuracy and number of clauses)



```
Accuracy: 0.9473684210526315
Max tree depth: 3
Num leaf nodes: 5

If/then clauses: 8
Decisions: 5
```

- Fewest clauses (ignoring accuracy)

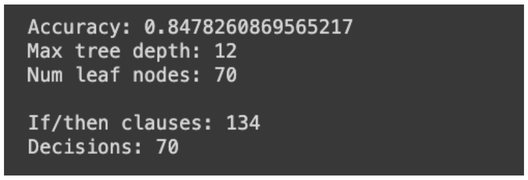


```
Accuracy: 0.8947368421052632
Max tree depth: 1
Num leaf nodes: 2

If/then clauses: 2
Decisions: 2
```

The best result was with a maximum depth of 3 and a maximum number of leaf nodes of 5. This not only reduced the number of if/then clauses from 30 to 8 and reduced the number of decisions from 16 to 5, but also resulted in a higher accuracy than my original tree. For this dataset, maximum leaf nodes was the more important strategy in reducing if/then clauses.

b) I then tested my same strategies on two more binary classification datasets.
Dataset #2: Australian credit (binary classification)
Original tree:

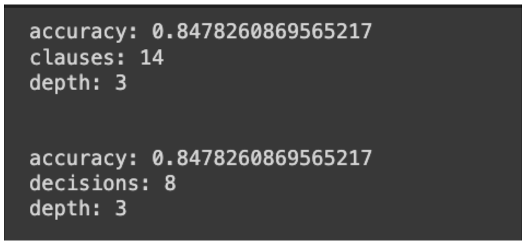


```
Accuracy: 0.8478260869565217
Max tree depth: 12
Num leaf nodes: 70

If/then clauses: 134
Decisions: 70
```

Strategies employed and best result:

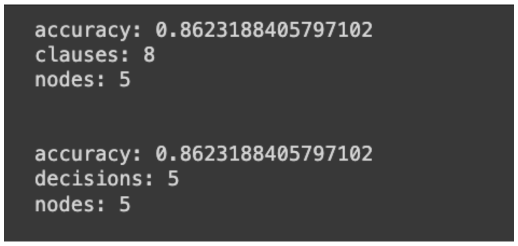
- Tune max_depth of tree



```
accuracy: 0.8478260869565217
clauses: 14
depth: 3


accuracy: 0.8478260869565217
decisions: 8
depth: 3
```

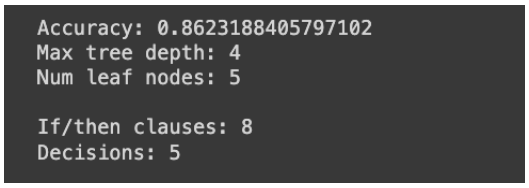
- Tune max_leaf_nodes of tree



```
accuracy: 0.8623188405797102
clauses: 8
nodes: 5


accuracy: 0.8623188405797102
decisions: 5
nodes: 5
```

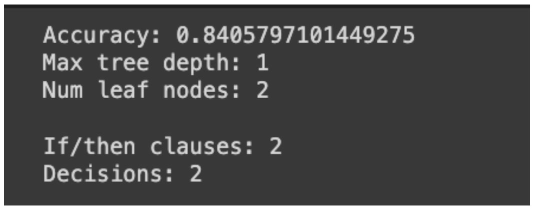
- Best result overall (prioritizing both accuracy and number of clauses)



```
Accuracy: 0.8623188405797102
Max tree depth: 4
Num leaf nodes: 5

If/then clauses: 8
Decisions: 5
```

- Fewest clauses (ignoring accuracy)

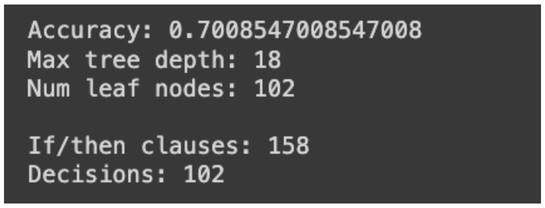


```
Accuracy: 0.8405797101449275
Max tree depth: 1
Num leaf nodes: 2

If/then clauses: 2
Decisions: 2
```

For this dataset, the best result was with a maximum depth of 4 (set to “None”) and a maximum number of leaf nodes of 5. For this dataset, maximum leaf nodes was the more important strategy in reducing if/then clauses. In fact, restricting maximum depth to 3 (which gave best results when only looking at maximum depth) when setting maximum leaf nodes to 5 actually gave worse results than the current best.

Dataset #3: Australian credit (binary classification)
Original tree:

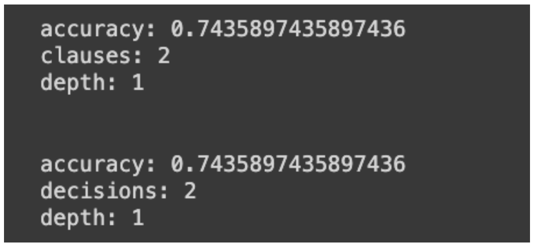


```
Accuracy: 0.7008547008547008
Max tree depth: 18
Num leaf nodes: 102

If/then clauses: 158
Decisions: 102
```

Strategies employed and best result:

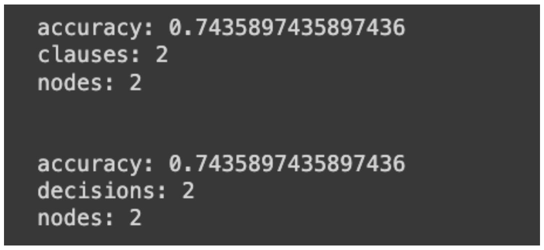
- Tune max_depth of tree



```
accuracy: 0.7435897435897436
clauses: 2
depth: 1


accuracy: 0.7435897435897436
decisions: 2
depth: 1
```

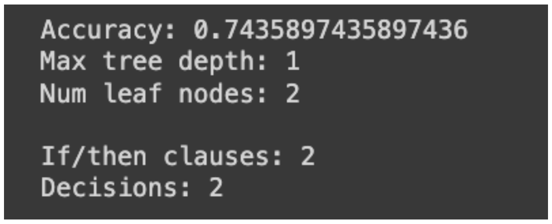
- Tune max_leaf_nodes of tree



```
accuracy: 0.7435897435897436
clauses: 2
nodes: 2


accuracy: 0.7435897435897436
decisions: 2
nodes: 2
```

- Best result overall and fewest clauses (same)

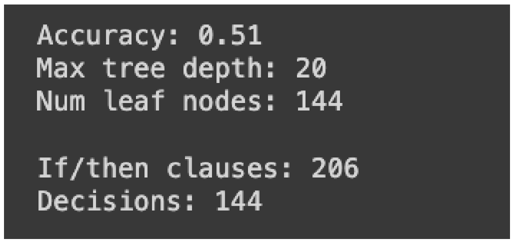


```
Accuracy: 0.7435897435897436
Max tree depth: 1
Num leaf nodes: 2

If/then clauses: 2
Decisions: 2
```

For this dataset, the best result was actually achieved with a maximum depth of 1 and a maximum number of leaf nodes of 2 (the most extreme case, minimum number of if/then clauses). Limiting the tree to only 2 clauses (and hence 2 decisions) also increased the accuracy.

c) Finally, I generated a completely random dataset and performed my strategies on it as well.
Dataset #4: randomly generated (binary classification)
Original tree:

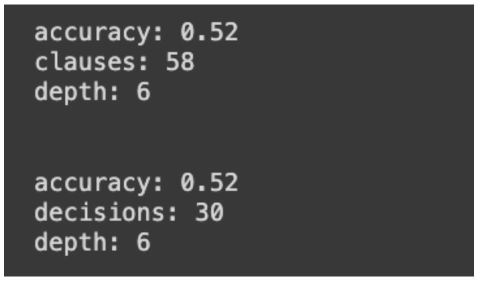


```
Accuracy: 0.51
Max tree depth: 20
Num leaf nodes: 144

If/then clauses: 206
Decisions: 144
```

Strategies employed and best result:

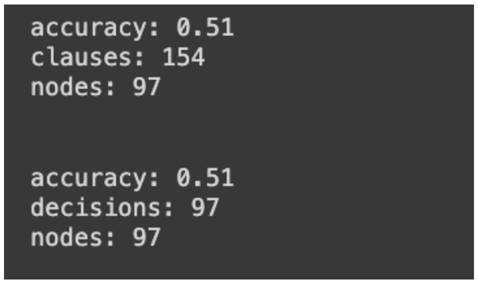
- Tune max_depth of tree



```
accuracy: 0.52
clauses: 58
depth: 6


accuracy: 0.52
decisions: 30
depth: 6
```

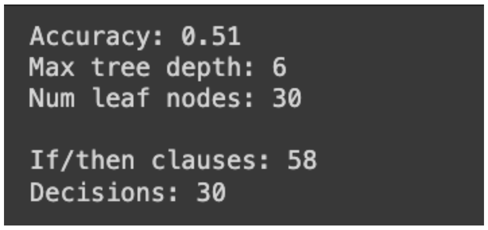
- Tune max_leaf_nodes of tree



```
accuracy: 0.51
clauses: 154
nodes: 97


accuracy: 0.51
decisions: 97
nodes: 97
```

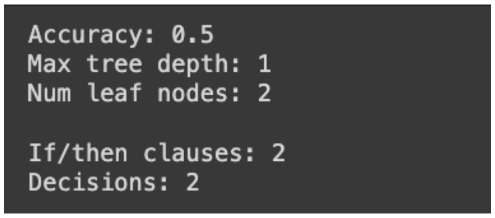
- Best result overall (prioritizing both accuracy and number of clauses)



```
Accuracy: 0.51
Max tree depth: 6
Num leaf nodes: 30

If/then clauses: 58
Decisions: 30
```

- Fewest clauses (ignoring accuracy)



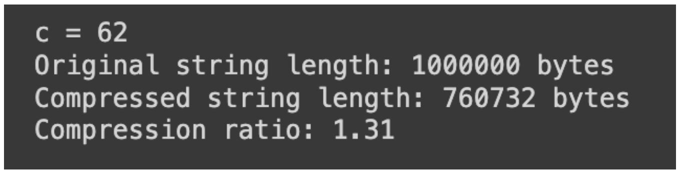
```
Accuracy: 0.5
Max tree depth: 1
Num leaf nodes: 2

If/then clauses: 2
Decisions: 2
```

In our current simulation on random data, we are training on a random train dataset and testing on a random test dataset (two different sets of data). In this case, you could possibly need an infinite number of if/then statements. If we were to instead test and train on the same dataset, we would need up to the *number of instances* of if/then statements to be able to achieve 100% accuracy (one if/then statement per instance).

Exercise 6.3:

a) Output:

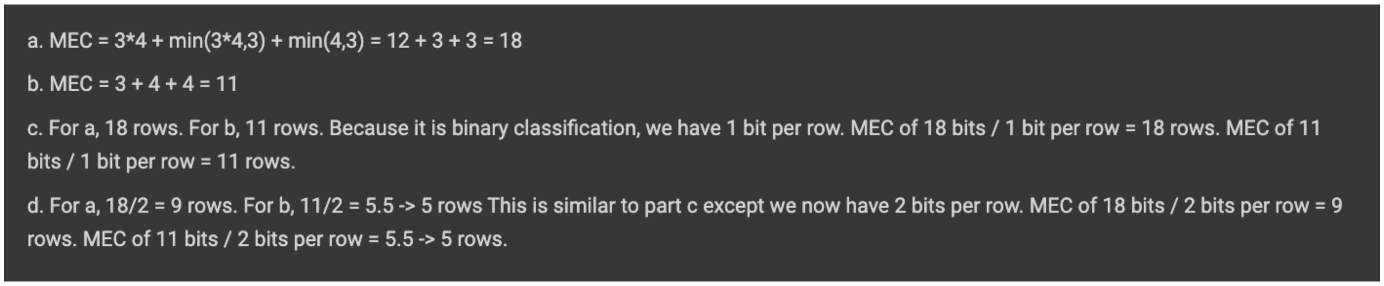


```
c = 62
Original string length: 1000000 bytes
Compressed string length: 760732 bytes
Compression ratio: 1.31
```

b) The expected compression ratio should be between 1 and 1.0164 (=62/61). This is because G for lossless compression is defined as 1 <= G <= (c/c-1), derived from our book for string compression). In this case, c is the number of unique ascii characters used to generate this string. G should not be less than 1 because in that case, it would indicate no compression.

Chapter 8

Exercise 8.1:



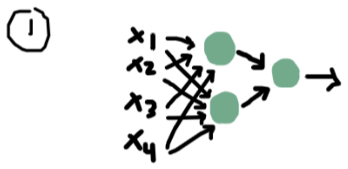
a. MEC = 3*4 + min(3*4,3) + min(4,3) = 12 + 3 + 3 = 18

b. MEC = 3 + 4 + 4 = 11

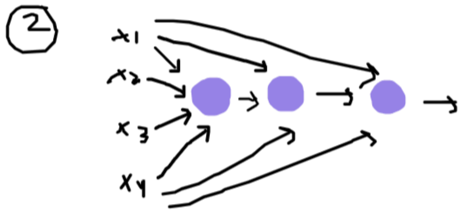
c. For a, 18 rows. For b, 11 rows. Because it is binary classification, we have 1 bit per row. MEC of 18 bits / 1 bit per row = 18 rows. MEC of 11 bits / 1 bit per row = 11 rows.

d. For a, 18/2 = 9 rows. For b, 11/2 = 5.5 -> 5 rows This is similar to part c except we now have 2 bits per row. MEC of 18 bits / 2 bits per row = 9 rows. MEC of 11 bits / 2 bits per row = 5.5 -> 5 rows.

Exercise 8.2:



$$(4+1)\cdot 2 + \min(3,2) = 5\cdot 2 + 2 = 10 + 2 = 12$$

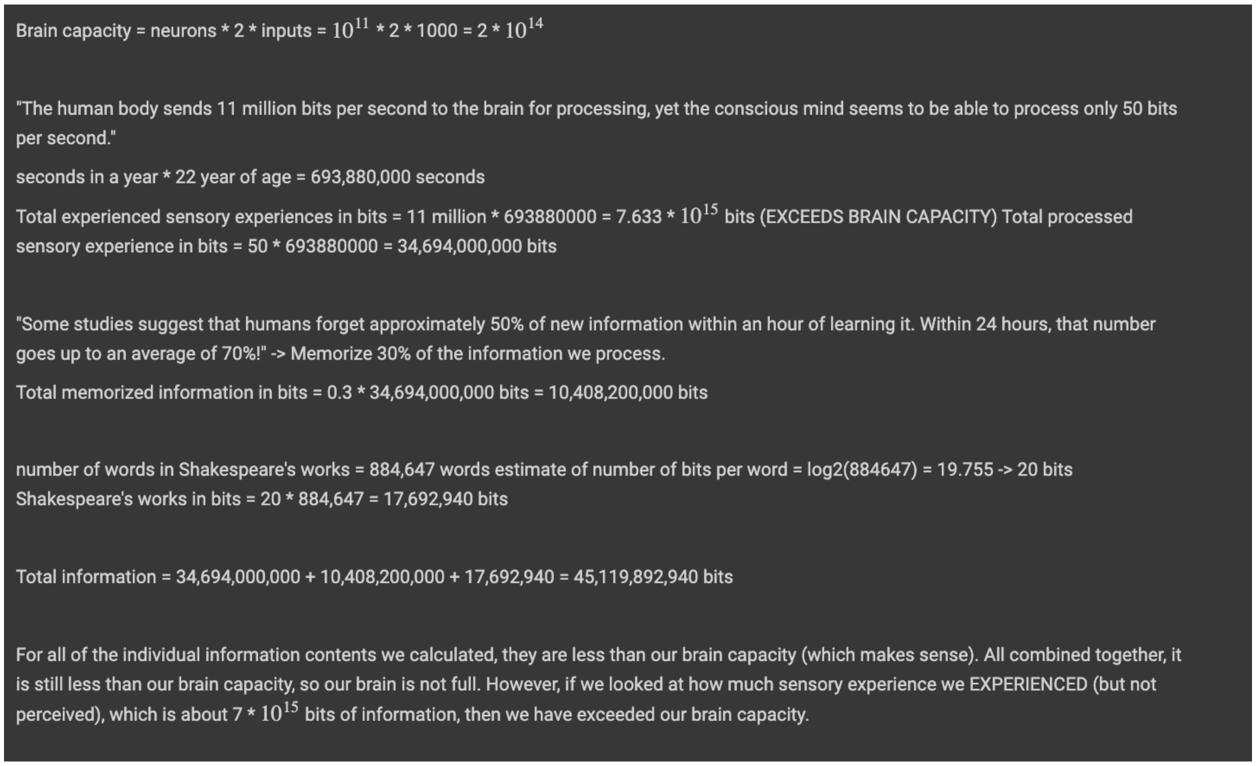


$$MEC = (4+1) + (3+1) + (3+1) = 5 + 4 + 4 = 13$$

Because these two architectures have MECs **equal to or greater than 12**, they can guarantee to memorize the training data of 12-instance binary classification problem of 4 dimensional inputs (12 rows of data).

Exercise 8.4:

a)



Brain capacity = neurons * 2 * inputs = 10^{11} * 2 * 1000 = 2 * 10^{14}

"The human body sends 11 million bits per second to the brain for processing, yet the conscious mind seems to be able to process only 50 bits per second."

seconds in a year * 22 year of age = 693,880,000 seconds

Total experienced sensory experiences in bits = 11 million * 693880000 = 7.633 * 10^{15} bits (EXCEEDS BRAIN CAPACITY) Total processed sensory experience in bits = 50 * 693880000 = 34,694,000,000 bits

"Some studies suggest that humans forget approximately 50% of new information within an hour of learning it. Within 24 hours, that number goes up to an average of 70%!" -> Memorize 30% of the information we process.

Total memorized information in bits = 0.3 * 34,694,000,000 bits = 10,408,200,000 bits

number of words in Shakespeare's works = 884,647 words estimate of number of bits per word = log2(884647) = 19.755 -> 20 bits
Shakespeare's works in bits = 20 * 884,647 = 17,692,940 bits

Total information = 34,694,000,000 + 10,408,200,000 + 17,692,940 = 45,119,892,940 bits

For all of the individual information contents we calculated, they are less than our brain capacity (which makes sense). All combined together, it is still less than our brain capacity, so our brain is not full. However, if we looked at how much sensory experience we EXPERIENCED (but not perceived), which is about 7 * 10^{15} bits of information, then we have exceeded our brain capacity.

b) (See jupyter notebook for Algo8 code for multi class)
Explanation: For this, I was not sure if the question was asking for multiple sets of binary classes or multi-class, so I implemented both. At a high-level, to implement the former, I put *memorize()* in a for-loop that iterates over all sets of binary class labels and added the MECs together. For the latter, the MEC should not change going from binary to more classes. I explicitly expanded the last part of the *return* statement (*min()*) to reflect the underlying computation for multi-class.

c) (See jupyter notebook for Algo8 code for regression)
Explanation: I implemented this by breaking down the continuous output space of the regression into discrete intervals. If an output lies in an interval, it is considered to be the same label (if not, threshold increases).

Chapter 9

Exercise 9.1:

Part 1: Train CNN and experiment blindly (referenced Pytorch's CNN tutorial)

I experimented with learning rate, number of linear layers, and convolutional kernel sizes to see how different values would affect performance (best/final chosen setting bolded).

Learning rate

- 0.0001 -> 46% accuracy
- **0.001 -> 54% accuracy**
- 0.01 -> 26% accuracy

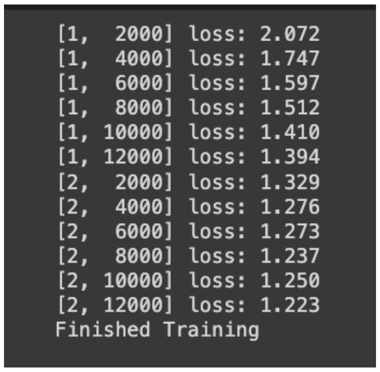
Linear layers

- **2 layers -> 55%**
    - self.fc1 = nn.Linear(400, 120)
    - self.fc2 = nn.Linear(120, 10)
- 3 layers -> 53%
    - self.fc1 = nn.Linear(400, 120)
    - self.fc2 = nn.Linear(120, 84)
    - self.fc3 = nn.Linear(84, 10)
- 4 layers -> 54%
    - self.fc1 = nn.Linear(400, 120)
    - self.fc2 = nn.Linear(120, 84)
    - self.fc3 = nn.Linear(84, 30)
    - self.fc4 = nn.Linear(30, 10)

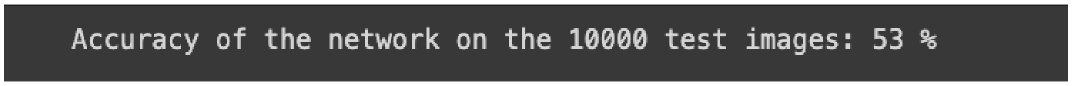
Convolutional kernel sizes

- 3 -> 54%
- **5 -> 54%**
- 7 -> 48%

Accuracy after training and experimenting with hyperparameters:



```
[1,  2000] loss: 2.072
[1,  4000] loss: 1.747
[1,  6000] loss: 1.597
[1,  8000] loss: 1.512
[1, 10000] loss: 1.410
[1, 12000] loss: 1.394
[2,  2000] loss: 1.329
[2,  4000] loss: 1.276
[2,  6000] loss: 1.273
[2,  8000] loss: 1.237
[2, 10000] loss: 1.250
[2, 12000] loss: 1.223
Finished Training
```



```
Accuracy of the network on the 10000 test images: 53 %
```

Part 2: Apply measurements

- Calculated necessary MEC of the linear layers based on the information passed to it by the previous convolutional layers

  Batch_inputs / (compression ratio) = input to/MEC of linear layers

  Batch_inputs = batch_size * in_channels * width * height = 4 * 3 * 32 * 32 = 12288

  Definition 9.1 (Convolutional Layer Generalization)

  According to:

  $$G_{conv} = \frac{(\#\ inputs) \times (input\ height) \times (input\ width) \times (input\ channels)}{(\#\ outputs) \times (output\ height) \times (output\ width) \times (output\ channels)}. \quad (9.1)$$

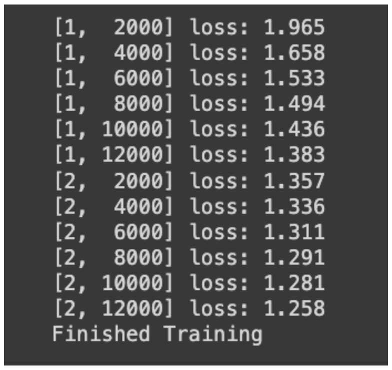
  ...

  Compression ratio = Gtotal = Gconv1 * Gconv2 = $\frac{4 \cdot 3 \cdot 32 \cdot 32}{4 \cdot 6 \cdot 14 \cdot 14} \times \frac{4 \cdot 6 \cdot 14 \cdot 14}{4 \cdot 16 \cdot 5 \cdot 5}$ = 7.68

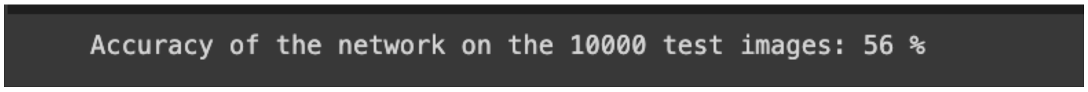
  inputs to linear layers = 12288 / 7.68 = **1600**
  This means that the MEC of my linear layer must be at least 1600.

- With the 3 layer architecture in my code, the MEC of the linear layers is (400+1)*120 + min((120+1)*84, 120) + min((84+1)*10, 84) = 48324. This is far bigger than 1600, which is unnecessary.
- Taking this into account, we will adjust the model again to reduce this MEC to a smaller value. This reduces the hyperparameter search space to the number of linear layers. Given the input size of 400 to the linear layer and output of 10 classes, **we can reduce the 3 linear layers down to a single *nn.Linear(400, 10)* layer** to minimize MEC (while still greater than 1600).
- With this change, our new MEC would simply be 400*10 = 4000, which is much closer to 1600.

Making this change, I got my final accuracy of **56%**, as opposed to my previous accuracy of 53%.



```
[1,  2000] loss: 1.965
[1,  4000] loss: 1.658
[1,  6000] loss: 1.533
[1,  8000] loss: 1.494
[1, 10000] loss: 1.436
[1, 12000] loss: 1.383
[2,  2000] loss: 1.357
[2,  4000] loss: 1.336
[2,  6000] loss: 1.311
[2,  8000] loss: 1.291
[2, 10000] loss: 1.281
[2, 12000] loss: 1.258
Finished Training
```



```
Accuracy of the network on the 10000 test images: 56 %
```