2)  How are you defining your loss function when training your model?

1. model.compile( optimizer = 'adam', loss = 'categorical\_crossentropy', metrics = ['accuracy'] )
2. we defined our optimizer as the Adam optimization algorithm, which is an optimizer that can be used to update the networks weights iterative based on training data
   1. adam is different from the classical stochastic gradient descent (maintains a single learning rate for all weight updates and it does not change during traing). Adam instead of only adapting the parameter learning rate based on the average first moment, it also adapts the average of the second moments of the gradients (uncentered variance)
3. we set out loss function as categorical cross entropy. This loss function is suitable for multi-class classification models where there are two or more output labels, which in our case we have 4 different labels for the 4 different directions in the two agents. The output label would be assigned as one-hot category encoding value which is the form of 0 and 1s. This output label (if as integer form) is converted into categorical encoding using keras.
4. Lastly the metrics we used is ‘accuracy,’ which according to keras it would be converted to BinaryAccuracy, CategoricalAccuracy, and SparseCategoricalAccuracy. The accuracy would be converted based on the loss function and the model output shape. Thus in our case the accuracy would be converted to CategoricalAccuracy.

3)  In training, how many episodes on how many different gridworlds were necessary to get good performance of your model on the training data?

* 1. Project1 Dense:
  2. For the first agent, we first flattened out data input of dim x dim
  3. Used two dense layers with units of 32 and 16.
  4. Defined probabilities with the softmax() function by inputing the resulting output from the application of two dense layers (with activation of relu)
  5. Then the result would be compiled to a model with adam optimizer, categorical cross entropy as loss, and accuracy metric.
  6. The number of epoch the allowed us to find the optimal performance of the model on training data was 3 epoch.

1. The result:  
   Epoch 1/3 26983/26983 [==============================] - 80s 3ms/step - loss: 0.1386 - accuracy: 0.9425 - val\_loss: 0.1437 - val\_accuracy: 0.9416 Epoch 2/3 26983/26983 [==============================] - 88s 3ms/step - loss: 0.1337 - accuracy: 0.9449 - val\_loss: 0.1355 - val\_accuracy: 0.9443 Epoch 3/3 26983/26983 [==============================] - 89s 3ms/step - loss: 0.1295 - accuracy: 0.9465 - val\_loss: 0.1343 - val\_accuracy: 0.9460
   1. Project 2 Dense:
   2. this is different compared with project 1 agent.
   3. We have two inputs for the model to train
   4. The first input: is the maze kb.
   5. The second input is the ‘inference’ kb:
      1. The inference kb has all grids defined as ‘-1,’ as the agent progressing in the maze run, the inferences made in projects would change the grids value that ranges from 0-8 with each value representing an inferred information
   6. We concatenated the two kb data as one, reshaped it as (num of maze grids, 2, dimension, dimension). We then flattened the newly concated array, during the buildModel() function. Then the flattened array would undergo the process similar to the Dense architecture.
   7. Due to the amount of data for the agent3, we had 4 dense layers (100, 40, 20, 10), while maintaining the same process of building the model as agent 1.
   8. The Epoch for agent 3’s model agent, would be increased to 1000.

This was surprising, we made the agent1’s epoch to 3 to avoid overfitting, but for agent3’s epoch needed to be increased to 1000 in order for the model to reach loss: 0.3068 - accuracy: 0.8487

* 1. The validation prediction of the model on test data is:

1. Epoch 1/10
2. 313/313 [==============================] - 2s 5ms/step - loss: 0.3007 - accuracy: 0.8565 - val\_loss: 6.8039 - val\_accuracy: 0.4252
3. Epoch 2/10
4. 313/313 [==============================] - 2s 6ms/step - loss: 0.2905 - accuracy: 0.8561 - val\_loss: 6.8780 - val\_accuracy: 0.4212
5. Epoch 3/10
6. 313/313 [==============================] - 2s 5ms/step - loss: 0.3009 - accuracy: 0.8545 - val\_loss: 6.7852 - val\_accuracy: 0.4226
7. Epoch 4/10
8. 313/313 [==============================] - 1s 5ms/step - loss: 0.3458 - accuracy: 0.8370 - val\_loss: 6.3415 - val\_accuracy: 0.4253
9. Epoch 5/10
10. 313/313 [==============================] - 2s 5ms/step - loss: 0.2933 - accuracy: 0.8557 - val\_loss: 6.5099 - val\_accuracy: 0.4301
11. Epoch 6/10
12. 313/313 [==============================] - 2s 5ms/step - loss: 0.2778 - accuracy: 0.8584 - val\_loss: 6.8972 - val\_accuracy: 0.4208
13. Epoch 7/10
14. 313/313 [==============================] - 1s 4ms/step - loss: 0.2834 - accuracy: 0.8578 - val\_loss: 7.0093 - val\_accuracy: 0.4252
15. Epoch 8/10
16. 313/313 [==============================] - 1s 5ms/step - loss: 0.3155 - accuracy: 0.8488 - val\_loss: 6.9669 - val\_accuracy: 0.4245
17. Epoch 9/10
18. 313/313 [==============================] - 2s 5ms/step - loss: 0.3056 - accuracy: 0.8534 - val\_loss: 6.8301 - val\_accuracy: 0.4293
19. Epoch 10/10
20. 313/313 [==============================] - 1s 4ms/step - loss: 0.3207 - accuracy: 0.8502 - val\_loss: 6.8984 - val\_accuracy: 0.4272
21. <keras.callbacks.History at 0x7ff53e26c290>
    1. CNN agent1:
       1. The structure data conversion of agent 1’s CNN is similar to that of the data conversion in Dense layer architecture
       2. It is the modelbuild() function that makes the difference
       3. In the modelbuild(), agent 1’s CNN model builder has one layer of the Conv2D() with filters (filter=16), kernel size of 3, activation of relu and input of (dim, dim, 1)
       4. Then the result would be flattened to go through a dense layer of 4
       5. The compilation of optimizer, loss, and metric would be adam, categorical cross entropy, and accuracy.

The Epoch of the model is only 2, and acquired loss: 0.2683 - accuracy: 0.9065

While the validation between the test was loss: 0.1537 - accuracy: 0.9520 - val\_loss: 0.1812 - val\_accuracy: 0.9462

* 1. For CNN agent3:
     1. The structure of the agent 3’s CNN was similar to agent 1’s CNN build model function.
     2. But we increased the epoch to 1000 and the number of conv2d() is three and then flatten the results to undergo one dense layer.
     3. Then the compilation of the optimizer, loss, and metric would be adam, categorical entropy, and accuracy.

The training: loss: 0.2516 - accuracy: 0.8796

The validation: Epoch 1/10

313/313 [==============================] - 2s 6ms/step - loss: 0.2480 - accuracy: 0.8796 - val\_loss: 7.2299 - val\_accuracy: 0.4141

Epoch 2/10

313/313 [==============================] - 2s 6ms/step - loss: 0.2475 - accuracy: 0.8808 - val\_loss: 7.2102 - val\_accuracy: 0.4120

Epoch 3/10

313/313 [==============================] - 2s 6ms/step - loss: 0.2493 - accuracy: 0.8806 - val\_loss: 7.3105 - val\_accuracy: 0.4059

Epoch 4/10

313/313 [==============================] - 2s 6ms/step - loss: 0.2559 - accuracy: 0.8775 - val\_loss: 7.2973 - val\_accuracy: 0.4096

Epoch 5/10

313/313 [==============================] - 2s 8ms/step - loss: 0.2599 - accuracy: 0.8739 - val\_loss: 7.0886 - val\_accuracy: 0.4074

Epoch 6/10

313/313 [==============================] - 2s 6ms/step - loss: 0.2676 - accuracy: 0.8769 - val\_loss: 7.0862 - val\_accuracy: 0.4187

Epoch 7/10

313/313 [==============================] - 2s 5ms/step - loss: 0.2950 - accuracy: 0.8673 - val\_loss: 7.2993 - val\_accuracy: 0.4169

Epoch 8/10

313/313 [==============================] - 2s 8ms/step - loss: 0.2849 - accuracy: 0.8708 - val\_loss: 7.3954 - val\_accuracy: 0.4148

Epoch 9/10

313/313 [==============================] - 2s 8ms/step - loss: 0.2517 - accuracy: 0.8777 - val\_loss: 7.3782 - val\_accuracy: 0.4099

Epoch 10/10

313/313 [==============================] - 2s 6ms/step - loss: 0.2495 - accuracy: 0.8766 - val\_loss: 7.2909 - val\_accuracy: 0.4101

<keras.callbacks.History at 0x7fe90b50dc90>

4)  How did you avoid overfitting? Since you want the ML agent to mimic the original agent, should you avoid overfitting?

1. We avoided overfitting by increasing our data, and ending our training early by cutting down the epoch and the complexity of the modelbuild() function.
2. In the beginning we only trained 5000 data for both agents, but the training model’s came out to be repeatedly going in one direction without changing it, and the probability of the prediction output showed that the other three directions had no chance of being chosen before that one direction.
3. Thus we increased our data by 10000, to 15000 for agent1 and by 3000 for agent 3 to 8000.
4. And we had decreased epoch numbers to ‘end our training’ early as suppose to our previous 1000 epochs and 4 dense layers for agent 1 and 3, and 4 CONV2D() and 2 dense layer for CNN agent 1 and 3.
5. The model’s prediction was no longer repeatedly only one direction, the other three directions have a higher probability of happening.

5)  How did you explore the architecture space, and test the different possibilities to find the best architecture?

1. In the beginning we were able to produce a model with a loss of below 0.2, and a accuracy rate of above 0.95 by brute force of epoch iterations of over 1000.
2. When we applied the fit() function to compare the training data and test data with the model, the difference returned low val\_loss (below 0.3) and high val\_accuracy (above 0.8).
3. Unfortunately, this is not the best architecture. The model only moved in one direction even though it already reached the border.
4. Thus we knew that the model was overfitting.
5. We could not brute force a result with high accuracy and low loss, thus we had to change our approach of the architecture.
6. We tried multiple dense layers, data conversion, and epochs. In the end we tried to decrease the amount of training, the layers of Dense and CONV2D used to create the model, and number of EPOCH iterations.
7. The accuracy and loss values were not as good as before, but the output of the model prediction showed outputs of different directions.

6)  Do you think increasing the size or complexity of your model would offer any improvements? Why or why not?

1. Increasing the size or complexity of our model actually had the opposite effect on the model’s prediction in imitating the project 1 and 2 agents. The agents move according to the Heuristic value, what information are present in the kb, and how the agent interpret the current knowledge base.
2. Had we increased the complexity and size of the model, it may be taking into account information and/or unseen factors in the maze and kb. Even though the model agent would be able to reach the goal, but since the objective of the project is for the model agent to imitate the behavior of the project 1 agent and project 2 agent.
3. Also by increasing the size or complexity of the model, it may take into account noise data, thus disrupting the models prediction. This is overfitting. By using a less complex, simple model, we can see, though a lower validation accuracy and high validation loss.
4. But the low validation accuracy and high validation loss are based on the test data we inputted. The test data are not the agents, only directions the agents made based on the iteration’s kb.
5. So even though the two validation values are lower than that of complex model, the simpler model is actually more similar to the agents.
6. The model agent can show a behavior more similar to the two project’s agents.
7. Thus increasing the size and or complexity of the model would not always offer improvements to the model agents. Instead, they would very like overfit the model agents and decrease the similarity of the model.