CS 237B: Principles of Robot Autonomy II
Problem Set 1

Name: Pei-Chen Wu
SUID:pcwu1023

01/21/2022

Problem 1

(i) CODE

(ii) Plot a heatmap of the optimal value function obtained by value iteration over the grid. The red arrows are aligned with the black arrows.

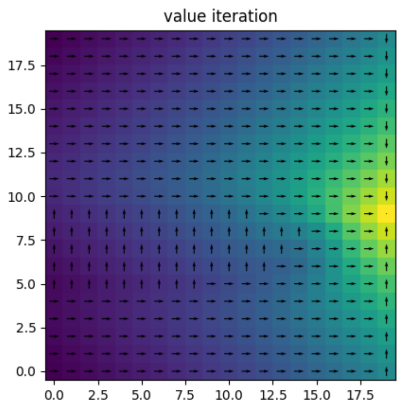


Figure 1: Value iteration heatmap

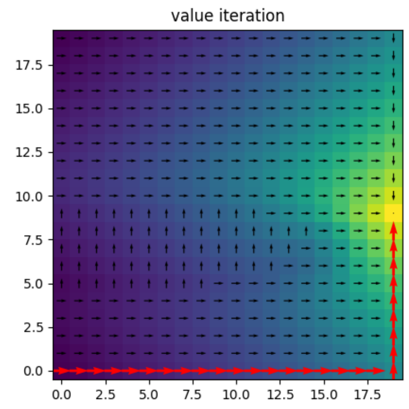


Figure 2: Optimal policy without Stochasticity

(iii) CODE

(iv) Plot the policy as a heatmap. Plot the simulated drone trajectory overlaid on the policy heatmap. The storm's influence is strongest at its center and decays farther from the center according to the equation. There is a slight chance that the storm will cause the drone to move in a uniformly random direction instead of being able to follow the optimal policy in reality.

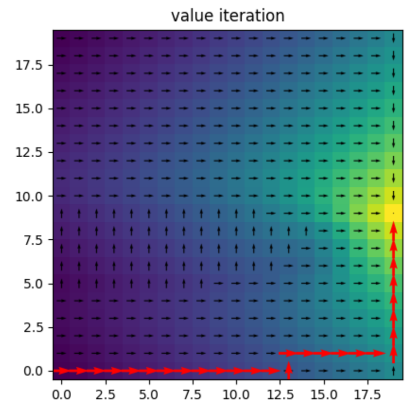


Figure 3: Optimal Policy with Stochasticity

(v) CODE

(vi) Bellman Equation:

$$Q(x_t, u_t) = \mathbb{E}\left[r_t(x,u) + \gamma \max_{\{u' \in U\}} Q(x_{t+1}, u')\right], \text{ if X is not terminal state.}$$

$$Q(x_t, u_t) = r_t(x,u), \text{ otherwise.}$$

(vii) CODE

(viii) CODE

(ix) Describe a dynamical system or a dataset situation in which using Q-learning could be easier than using value iteration.
Since Q-learning is model free RL algorithm, compares with value iteration which is model based needs transition probability. Q-learning will be easier for applications have unknown environment, such as self-driving car or mobile robot in an unknown environment, robot manipulator, stock trading, etc.

(x) A binary heatmap plot of where the approximate Q-network policy agrees with the value iteration optimal policy. Learning rate is playing an important role in this problem, lower learning rate will cause the result deviates from value iteration.

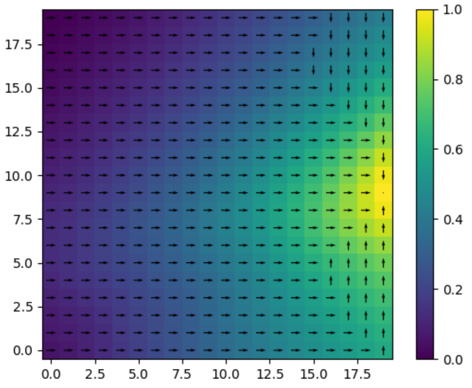


Figure 4: Q-Learning, lr = 1e-4

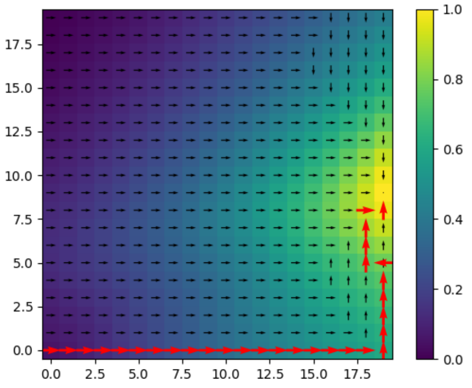


Figure 5: Q-Learning optimal policy w. stochasticity, lr = 1e-4

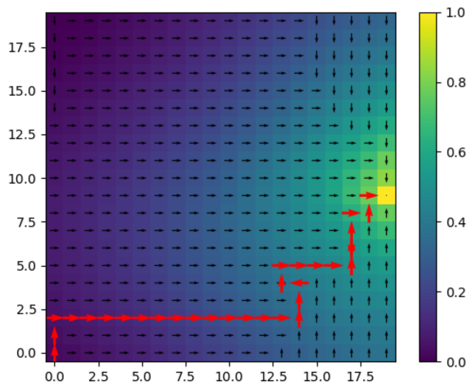


Figure 6: Q-Learning optimal policy w. stochasticity, lr = 1e-2

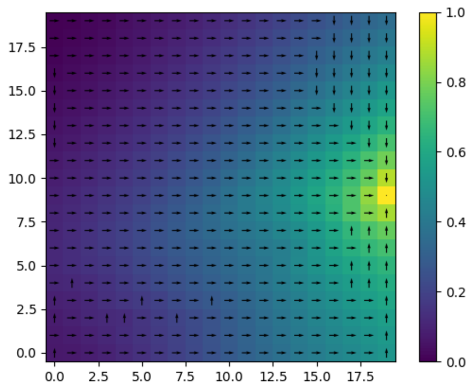


Figure 7: Q-Learning, lr = 1e-3

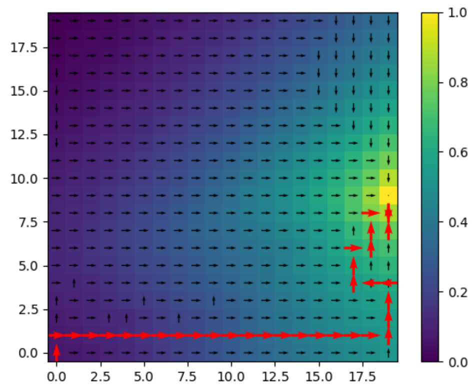


Figure 8: Q-Learning optimal policy w. stochasticity, lr = 1e-3

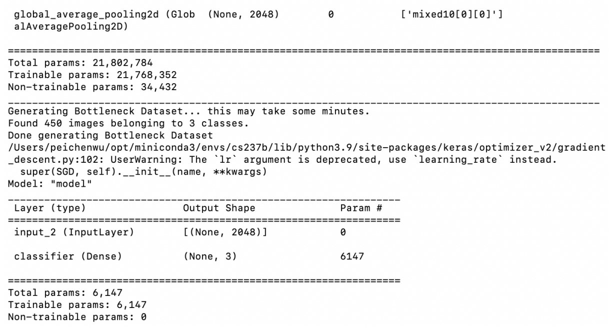
Problem 2

(i) CODE

(ii) CODE

(iii) CODE

(iv) bottleneck dim = (None, 2048) total params = 6147

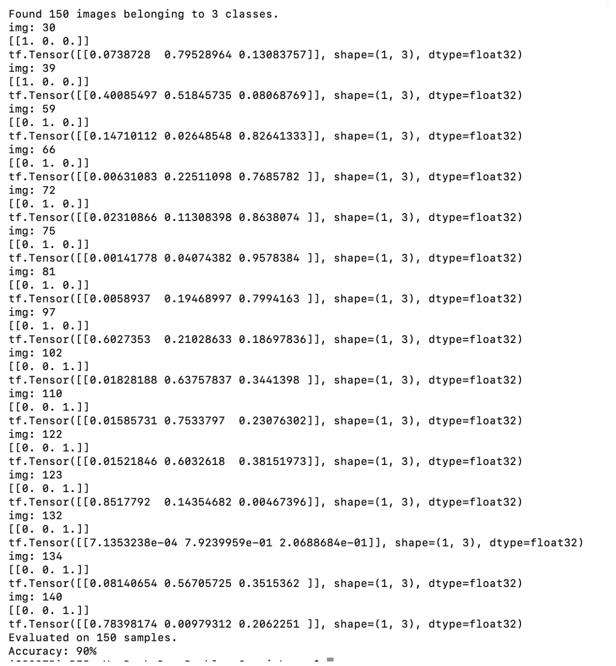


```
 global_average_pooling2d (Glob  (None, 2048)        0           ['mixed10[0][0]']
 alAveragePooling2D)

==================================================================================================
Total params: 21,802,784
Trainable params: 21,768,352
Non-trainable params: 34,432
__________________________________________________________________________________________________
Generating Bottleneck Dataset... this may take some minutes.
Found 450 images belonging to 3 classes.
Done generating Bottleneck Dataset
/Users/peichenwu/opt/miniconda3/envs/cs237b/lib/python3.9/site-packages/keras/optimizer_v2/gradient
_descent.py:102: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(SGD, self).__init__(name, **kwargs)
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 2048)]            0

 classifier (Dense)          (None, 3)                 6147

=================================================================
Total params: 6,147
Trainable params: 6,147
Non-trainable params: 0
```

Figure 9: retrain model

(v) Training the whole classifier from scratch will take much more time and usually dropout layers will be used to prevent overfitting problem. However, with the dropout we are also scaling our neurons. Basically, if it's 0.9 dropout keep probability we need to scale it by 0.9, which means we are getting 0.1 larger in the testing . Therefore, dropout layers can saturate the nodes which causes the non converging issue.

(vi) CODE



```
Found 150 images belonging to 3 classes.
img: 30
[[1. 0. 0.]]
tf.Tensor([[0.0738728  0.79528964 0.13083757]], shape=(1, 3), dtype=float32)
img: 39
[[1. 0. 0.]]
tf.Tensor([[0.40085497 0.51845735 0.08068769]], shape=(1, 3), dtype=float32)
img: 59
[[0. 1. 0.]]
tf.Tensor([[0.14710112 0.02648548 0.82641333]], shape=(1, 3), dtype=float32)
img: 66
[[0. 1. 0.]]
tf.Tensor([[0.00631083 0.22511098 0.7685782 ]], shape=(1, 3), dtype=float32)
img: 72
[[0. 1. 0.]]
tf.Tensor([[0.02310866 0.11308398 0.8638074 ]], shape=(1, 3), dtype=float32)
img: 75
[[0. 1. 0.]]
tf.Tensor([[0.00141778 0.04074382 0.9578384 ]], shape=(1, 3), dtype=float32)
img: 81
[[0. 1. 0.]]
tf.Tensor([[0.0058937  0.19468997 0.7994163 ]], shape=(1, 3), dtype=float32)
img: 97
[[0. 1. 0.]]
tf.Tensor([[0.6027353  0.21028633 0.18697836]], shape=(1, 3), dtype=float32)
img: 102
[[0. 0. 1.]]
tf.Tensor([[0.01828188 0.63757837 0.3441398 ]], shape=(1, 3), dtype=float32)
img: 110
[[0. 0. 1.]]
tf.Tensor([[0.01585731 0.7533797  0.23076302]], shape=(1, 3), dtype=float32)
img: 122
[[0. 0. 1.]]
tf.Tensor([[0.01521846 0.6032618  0.38151973]], shape=(1, 3), dtype=float32)
img: 123
[[0. 0. 1.]]
tf.Tensor([[0.8517792  0.14354682 0.00467396]], shape=(1, 3), dtype=float32)
img: 132
[[0. 0. 1.]]
tf.Tensor([[7.1353238e-04 7.9239959e-01 2.0688684e-01]], shape=(1, 3), dtype=float32)
img: 134
[[0. 0. 1.]]
tf.Tensor([[0.08140654 0.56705725 0.3515362 ]], shape=(1, 3), dtype=float32)
img: 140
[[0. 0. 1.]]
tf.Tensor([[0.78398174 0.00979312 0.2062251 ]], shape=(1, 3), dtype=float32)
Evaluated on 150 samples.
Accuracy: 90%
```

Figure 10: classification

(vii) CODE

(viii) In addition to filling out compute brute force classification, include the detection plot for your favorite image in catswithdogs.

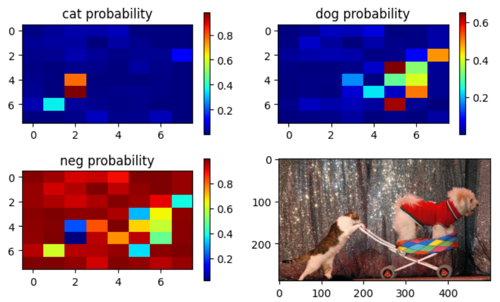


Figure 11: brute force classification

(ix) Find the output of the final convolutional layer (mixed 10). What operation does it feed into? How is the feature vector for the image as a whole computed from the feature vectors for each image region? The mixed 10 layer is a concatenate layer, which concate activation−85 (8,8,320), mixed9−1(8,8,768), concatenate−1(8,8,768) and activation−93(8,8,192). The feature vector for the image didn't work every well. In brute force classification, we use the image classifier on the segmented image, which is different from what we did in training, we used the whole cat or dog image in most of the data in the training dataset.

(x) CODE

(xi) Include in your writeup the detection plot for your favorite image in catswithdogs.

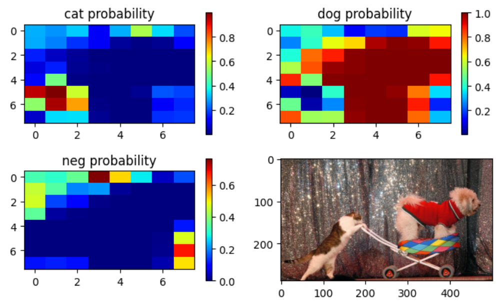


Figure 12: conv detector

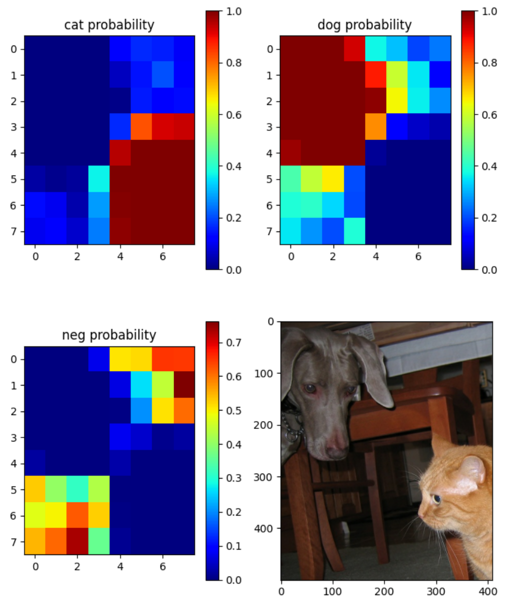


Figure 13: conv detector

(xii) CODE

(xiii) include in your write up the results on both a correctly and incorrectly classified image from test.

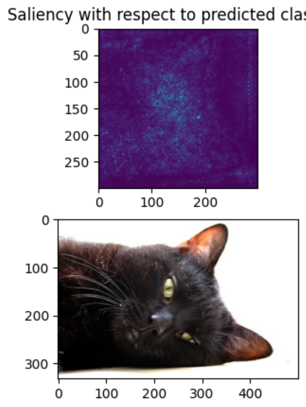


Figure 14: saliency - correct classification

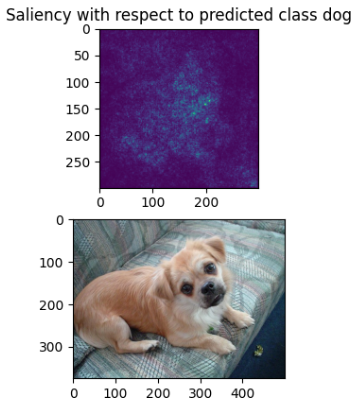


Figure 15: saliency - correct classification

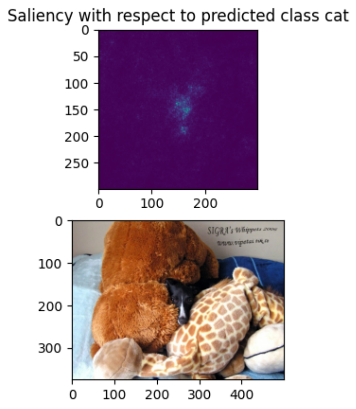


Figure 16: saliency - incorrect classification