Internship Assignment – 2016

Rishabh Malviya
(IIT Bombay)

Problem 2 (Common Assignment)

Question 1

Euler Angles
Advantage: Euler Angles are an especially intuitive way to represent rotations. They are ideal for describing single transformations from a fixed frame (as opposed to composing transformations).
Disadvantage: Using Euler Angles can lead to the problem of Gimbal Lock and leads to the restriction of the rotation about a single axis only.
Unit Quaternions
Advantage: These are computationally very efficient for use in code. They also don't face the problem of gimbal lock because their topological space of S^3 is a covering group of RP^3, the topological space of SO(3), the group of all rotations in 3-D space.
Disadvantage: There is often some ambiguity regarding the convention to use with quaternions - *(w,x,y,z)* or *(x,y,z,w)*
Rotation Matrices
Advantage: These are useful as a starting points for determining axis-angle representations and Euler Angles.
Disadvantage: These are the most expensive computationally (in terms of memory) - you have to store 9 numbers for a 3-parameter object.
Axis-Angle Representations
Advantage: They are very useful for the visual understanding of the rotation.
Disadvantage: While they only use three numbers to represent the rotation, you cannot compose two axis-angle rotations without the help of another representation such as rotation matrices or quaternions.

Question 2

R_a = 0.000000, 0.500000, -0.866025
-0.707107, 0.612372, 0.353553
0.500000, 0.866025, 0.000000

R_b = -0.866025, 0.500000, -0.000000
-0.000000, 0.000000, 1.000000
0.000000, 0.000000, 0.000000

Question 3

These quaternions follow the (x,y,z,w) convention. They are w.r.t. a fixed frame where the roll axis is the z-axis, the pitch axis is the y-axis and the yaw axis is the x-axis.
Each quaternion isn't unique to its corresponding rotation, because it's negative would represent the same rotation:

q_a = (0.461940,0.331414,-0.191342,0.800103)

q_b = (0.000000,0.000000,-0.500000,0.866025)

Question 4

The composed rotations are given by:

q_c = (0.234345,0.517982,-0.565758,0.597239)

q_d = (0.565758,0.056043,-0.565758,0.597239)

Question 5

q_e = (0.565758,0.056043,0.234345,0.788581)

q_f = (0.461940,0.331414,-0.191342,0.800103), which is the same as q_a

Problem 5 (Semantic Perception)

Question 1 (Machine Learning Concepts)

a) Classification and regression both try to find a function to map input data (usually vectors of features or raw data) to their appropraite outputs. The difference lies in what those outputs are.
   i. For **regression**, the range of the function is usually a continuum such as the set of real numbers or some other $\mathbf{F}^n$, where **F** denotes a field (examples are the set of real numbers, **R,** or the set of complex numbers, **C**).
   ii. In **classification** on the other hand, we want the output of our function to be one or the other class, which belongs to a set of pre-defined classes. The catch is that the classes do not have any inherent order, and this is what makes the task significantly different from regression. A common strategy is to define a mapping between intervals on the real line to the classes; this allows the task to be tackled just like in regression – the only change is in how we interpret the (real number) outputs of the system.

b) The role of the three subsets is explained below:
   i. **Training:** Every machine learning system eventually tries to create a 'model' which can faithfully represent the relationship between inputs and outputs that arise in some complex process (for example, the relationship between handwritten digits and the their corresponding numeric values they represent). These models are created using some data, which is usually a subset of the original dataset and is labeled as 'training' because it is used to 'train' the system (i.e., this is what the machine learning system 'learns' from).
   ii. **Testing:** In order to guage the performance of the system and the model it creates, a testing subset (which is completely disjoint from the training subset) is used to measure how well the predictions of the model and the actual outputs agree. There are many statistical measures used to quantify the performance of a system on the testing data such as the precision/recall values, F1 measure, rand index, etc.
   iii. **Validation:** This is very similar to a testing subset, but it is usually only useful when the system is to be deployed for use in real-world scenarios where it mus make predictions based on data that isn't 'pre-labelled'. For example, we may train a music transcription system on the recordings of a certain violin played by a certain violinist. But when the system is to be used by other violinists with different violins, we can't be sure that the system will adapt well to those changes. This is a common problem in machine learning called *overfitting.* A validation set is usually set aside to be sure that the system isn't overfitting to the training data; after the system is trained, the system's performance is measured on the validation set, where it shouldn't be significantly lower than it's performance on the training set itself. This process is usually made more robust through *cross-validation*, wherein the dataset is divided randomly into training and validation multiple times to be sure that the system isn't overfitting to any specific training subset.

c) It would be hard to comment on whether or not the problem is solved without knowledge of the details of how the “99% accuracy” was measured. An oft-overlooked detail in research of this kind is the dataset on which the accuracy is being reported – is it the training data itself, a separate testing subset, or real-world data collected by the researchers themselves? I observed this recently in some research papers which were reporting the accuracy of a neuromorphic hardware on a machine learning task. In this case though, if the species of bird being detected is rare (i.e., significantly different visually from any other species of bird), measuring performance on training data alone - though cross-validation with validation sets would be ideal - might be fine; the idea behind this is that any other real-world data (for example, a picture of another bird) would find it very hard to fool the system, since what the system is designed to detect is unique enough to minimize the chances of false positives as it is. But, if the accuracy is reported on the training data itself, it would be important to assess the quality and diversity of the training data - in terms of the various orientations of the bird, various lighting conditions, etc.

d) Most machine learning systems create models to represent the relationship between inputs and outputs that arise in complex processes. These models are created from data that is fed to the system. Often, the data being fed isn't pre-labeled with any output values. It is then the task of the system to discern patterns in the data, form representations of these patterns and develop the ability to detect them. This is what **unsupervised learning** algorithms aim to do. When the model is created using pre-labelled data, we have **supervised learning.** The advantage of unsupervised learning over supervised is primarily that datasets do not have to be pre-labelled (for example, in the semantic segmentation task, the very difficult task of doing an accurate pixel-wise labeling for creating a training dataset can be avoided). The advantage of supervised learning is that there are well-established methods to check for and avoid overfitting. Overfitting is difficult to detect in unsupervised learning systems – the patterns and representations formed by the system may arise out of a peculiarity of the dataset being trained on, and we can't know for sure that these patterns are irrelevant for other real-wold data.

e) The approach may not work very well, because the identity of what object a pixel belongs to is often dependent on the nearby pixels in the image. The only information a pixel can give us is the RGB (or intensity) values. These are very few features to base a classifier on. An improvement could be to make use of nearby pixels through a convolutional approach (as outlined in the F-CNN paper mentioned in the assignment). Another approach would be to group together similar pixels through segmentation algorithms and extract statistics from the resulting superpixels. The statistics extracted from the superpixels can give us many more features than just a single set of RGB or intensity values can. This is the approach I have used for my Semantic Labeling task (Question 2 of this Problem).

f) **Parameters** usually refer to the inter-neuron weights that are actually 'trained' in a network. **Hyperparameters** are the variables in a neural network that remain fixed during training. These are like properties of the network that are pre-defined by a user. Examples of hyperparameters are the learning rate, the number of hidden layers and the number of neurons in each of those layers, etc. A different set of

hyperparameters will essentially give you a different neural network.

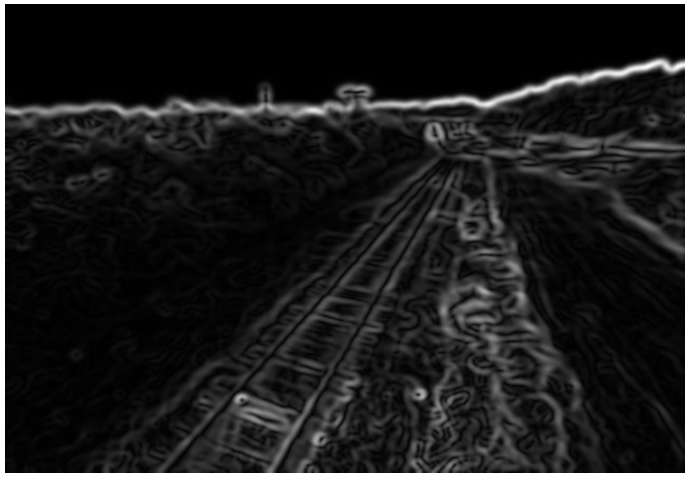
g) Hyperparamters cannot be trained using standard gradient descent. This follows quite obviously from the definition given in the previous question, but can also be understood by noting that the backpropagation algorithm ('standard' gradient descent in the case of deep neural networks) changes the values of the inter-neuron weights – which are continuous parameters – while hyperparameters like the number of neurons in a given layer can only assume whole number values. This distinguishes hyperparameters and parameters using reasoning similar to that used to distinguish regression from classification.

Question 2 (Semantic Labeling)

The entire pipeline can be divided into four parts. Each part has a corresponding source file, and the roles of each are explained below. Please note that the scripts must be run after cd-ing into the */scripts/* folder, and the individual executables must be run after cd-ing into the */build/* folder (after running CMake and Make):

1. **features_slic (Segmentation, Feature Vector Extraction):** The first task is to segment all the images into superpixels and then extract feature vectors from each of those superpixels. This task is done recursively for all the files using the *scripts/FV_Extraction* script.:
   1) As an input, this executable requires the image number ('0000' or '0480', refering to the left images in the */data/ml_task/* folder).
   2) It then performs a SLIC segmentation of the image, which is a very powerful technique that only uses the color and position information of the pixels.:


   3) Once we have the superpixels, the feature vectors for each superpixel are calculated and saved. These feature vectors are determined using properties like the average RBG values, Gaussian Gradients, Cornerness, etc. and are what get fed to the subsequent classifiers in the pipeline. The following image shows what the Gaussian Gradient property looks like:

2. **GT_SLICLabels_extraction (Extract Ground Truth Values and SLIC Labels):** This file extracts the ground truth labels (the desired outputs) for each feature vector using the labeled images. These are determined by taking the most frequent pixel-wise label occuring in each superpixel corresponding to each feature vector. The step is done for all of the training images using the *scripts/GT_SLICLabel_Extraction* script and the output of is stored in these folders:
   - *data/GT_vectors* - Each file in this folder lists the ground truth values for each superpixel in the form of a column vector - one vector per image, and one superpixel per entry.
   1) *data/SLICLabel_vectors* - This contains the labels of the superpixels (which are numbers). The reason I had to output this explicitly is because of a small glitch in the VIGRA SLIC segmentation implementation.
   2) *data/SLICSegmentation_arrays* - This contains arrays of the size of the images, wherein each pixel is labeled with the label of the superpixel it belongs to.

3. **kNN (k - Nearest Neighbours to Assign Labels to Feature Vectors) :** A particular training-testing division is chosen (37 images training, 13 images testing) and the feature vectors of the superpixels in the training images are used as reference points for the k-NN algorithm. The 5 nearest neighbors of the testing feature vectors are used to determine what their labels should be. One way to proceed is this:
   - The label of the feature vector is predicted by assigning scores to all of the possible labels of the 5 nearest neighbours against the inverse distances from those neighbours. That is, if α_j is the score for label 'j' and d_k denoted the distance to a nearest neighbor with label 'k':

$$\alpha_j = \frac{\sum_{k=1}^{K} \frac{1}{d_k} \delta_{kj}}{\sum_{k=1}^{K} \frac{1}{d_k}}$$

   - The label with the highest score is then chosen as the label of the feature vector.

   But, the algorithm used in this pipeline further transforms this vector of scores (called a 'confidence vector') using extra information extracted from the training vectors. It then predicts the label of the vector to be the one with the maximum score. The exact algorithm for this is explained below:
   1) The extraction of extra information from the training vectors begins by finding the 4 nearest neighbours for each of the training vectors (the first nearest neighbour would be the training vector itself, which is part of the set of reference points for the algorithm).
   2) These 4 nearest neighbours are used to determine the expected labels for the training vectors (using the naive algorithm described above). These expected labels are used with the real labels to create a confusion matrix for the training vectors. These are what the */data/Predictions_n/Performance_Evaluation/intermediateConfusionMatrix.csv* files hold.
   3) This confusion matrix is then transformed into a conditional probability matrix

where the (i,j) entry of the matrix denotes the probability that the actual label (GroundTruth value) is 'i', given that the kNN search predicted the label 'j'. These are what the */data/Predictions_n/Performance_Evaluation/condProb.csv* files hold.

$$v_{i|j} = Probability(groundTruth = i|prediction = j) = \frac{C_{i,j}}{\sum_{i'} C_{i',j}}$$

4) The kNN search is then run for the testing vectors and confidence vectors are obtained.
5) These confidence vectors are then transformed using the conditional probability matrix. That is, the ith entry of the confidence vector now becomes the dot product of the ith row of the conditional probability matrix with the original confidence vector.

$$v_i = \sum_{j \in \text{cat}} \alpha_j * v_{i|j}$$

6) Now, the index with the maximum value in the transformed confidence vector is assigned as its label.

NOTE: *The number of nearest neighbours to consider in the various parts of the algorithm can be varied, though in the preceding discussion we assumed fixed numerical values.*
NOTE: *To run the algorithm without the added modifications, comment the for loop starting at line 449 of the kNN.cpp file and uncomment the line just after it - line number 452.*

4. **eval (Quantify Pipeline Performance):** This script *evaluations* takes the predicted images and compares them pixel-wise with the original labeled (ground truth) images to calculate the accuracy (confusion matrix, precision, recall and F1) of the pipeline.

   Another approach is to make the comparison superpixel-wise, and there is a separate script called *evaluations_superpixel* for doing this. As would be expected, the reported accuracy is much higher for this superpixel-wise comparison; but it is not really a correct representation of the accuracy of the pipeline, as the final task is to label the pixels themselves.

   There was a minute increase in 'pixel-wise' performance thanks to the introduction of the explained modification in the kNN search algorithm. The left side image shows the results without the modification to the kNN algorithm, and the right side with: