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

```
Ra =   0.000000,  0.500000, -0.866025
      -0.707107,  0.612372,  0.353553
       0.500000,  0.866025,  0.000000


Rb =  -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 <u>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?</u> 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, <u>if the accuracy is reported on the training data itself, it would be important to assess the quality and diversity of the training data</u> - 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)

I have created the following pipeline to solve the task. It makes use of superpixels, which are separately extracted from the image:

1. Segement the pixels in the image into superpixels using SLIC segmentation algorithm. The system is designed to lable all pixels in a given superpixel with the same label.
2. Calculate feature vectors for each of the superpixels in the image using statistics such as the average and variance of RGB values, average and variance of Gaussian Gradients, etc. Append the mode of the disparity value in the superpixel to the vector.
3. Find the most frequent labeling of the pixels in a superpixel (in the corresponding labeled image) and append it to it's feature vector. We now have a training set for superpixels with labels given by this last component.
4. Now, 3/4 (37 images) of this set is chosen at random and used as seeds (reference data points) for a k Nearest Neighbours algorithm applied to the rest of 1/4 of the images (13 images).

I have tried to incorporate an extra algorithmic improvement to the basic kNN algorithm. Ideally, it would require the use of a validation set, but in this assignment I have re-used the training data for this. This improvement is loosely inspired by Kalman Filtering and very similar to Bayesian Classifiers. It is explained in detail in the *'Data Fusion'* section of my paper *'Neuromorphic Hardware Accelerated Adaptive Authentication System'*, which was published in the proceedings of the IEEE International Conference on Evolvable Systems a few days ago.

It tries to qualitatively capture the biases a classifier may have to certiain labels (outputs). Since kNN can give us up to 'k' predictions for a given superpixel's feature vector, we can come up with a 'prediction vector' denoting the system's prediciton for the superpixel as follows:

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

The index 'j' corresponds to the possible predictions (classes) into which the nearest neighbours fall. 'K' is the total number of nearest neighbours being considered. This is essentially a vector of values each corresponding to how 'strongly' the K-nearest neighbours together classify the vector into a certain category ('j').

Intuitively now, it makes sense to simply take the index with the maximum value and assign it to the superpixel. But, we can go a step further than this. We begin by calculating the predictions for the training data (ignoring the first neighbour, which would be itself) and create a confusion matrix on that basis. This confusion matrix captures the bias of the system. We now calculate some useful probabilities using this confusion matrix:

$$v_{i|j} = \text{Probability}(\text{groundTruth}=i \mid \text{prediction}=j) = C_{i,j}/\Sigma_{i'}C_{i',j}$$

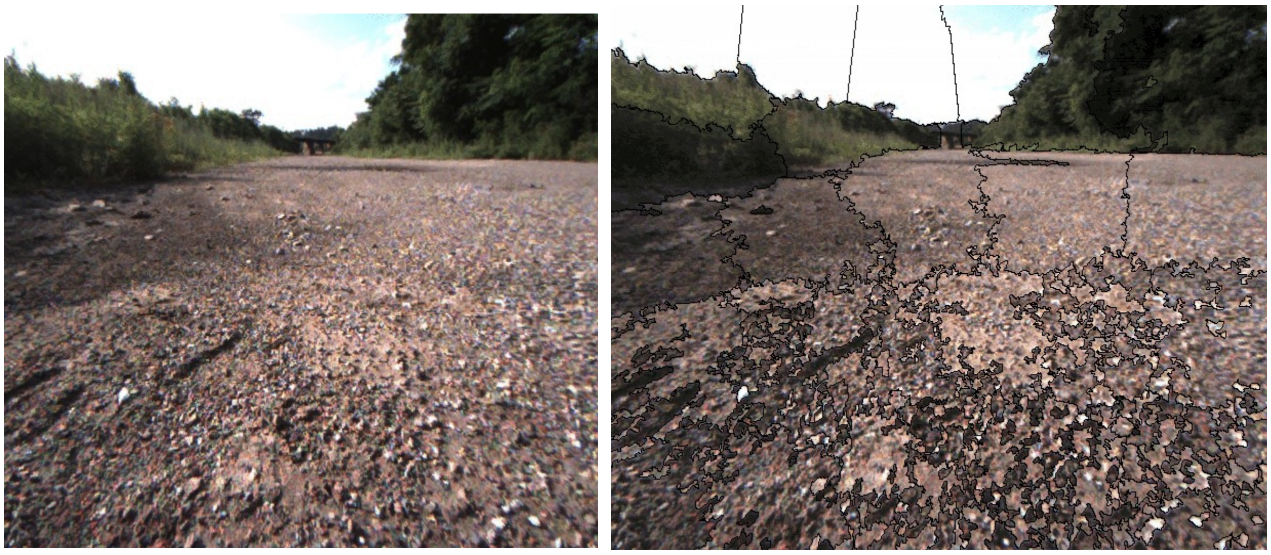
These can be used to futher refine the prediction vector we just got from the kNN classifier. BTW, these confusion matrices and conditional probabilities are stored for each case in the files *'intermediateConfusionMatrix.csv'* and *'condProb.csv'* in the *data/Prediction_** folders, respectively.

Now we refine the prediction vectors and get a new vector called the 'confidence vector' as follows:

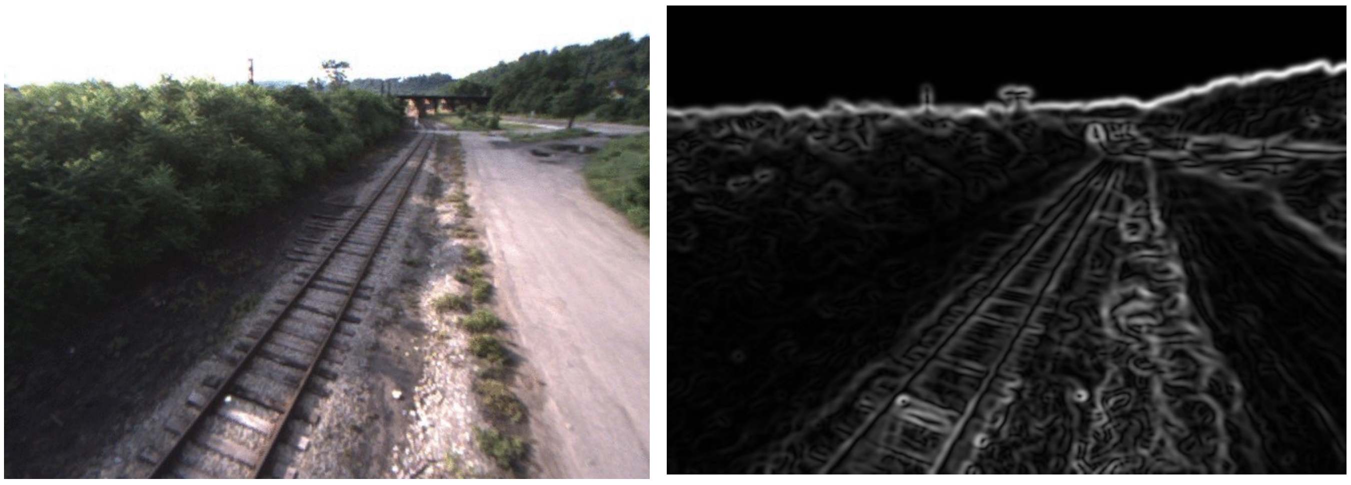
$$v_i = \sum_{j \in \mathrm{cat}} \alpha_j * v_{i|j}$$

In essence, this transformation takes us predictions by the classifier to ground truth values based on how reliable the classifier is for different values and the classifiers confidence in it's own predictions (through the 'prediction vector'). It is from this vector that the index with maximum value is taken and assigned to the superpixel.

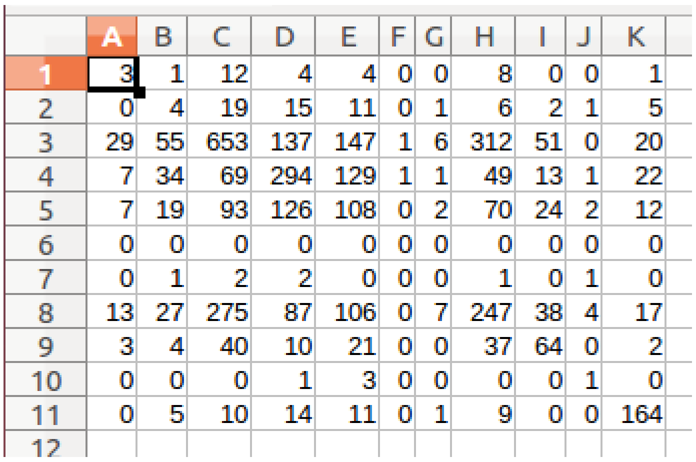
Image Segmentation:



Feature Extraction (Gaussian Gradients features):

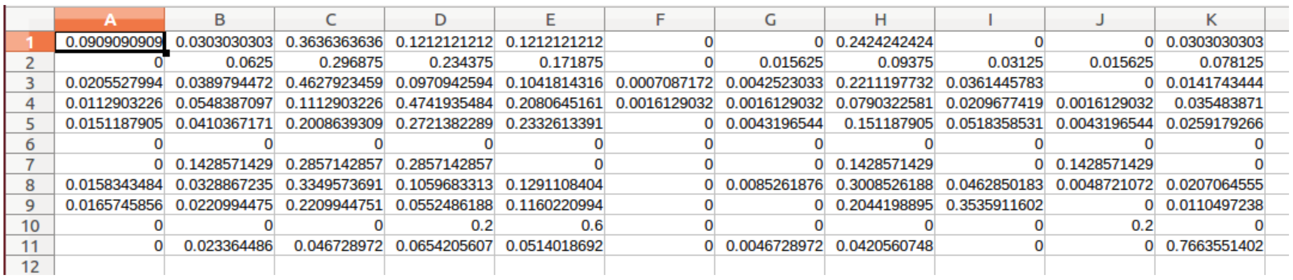


A sample (intermediate) confusion matrix is shown here:



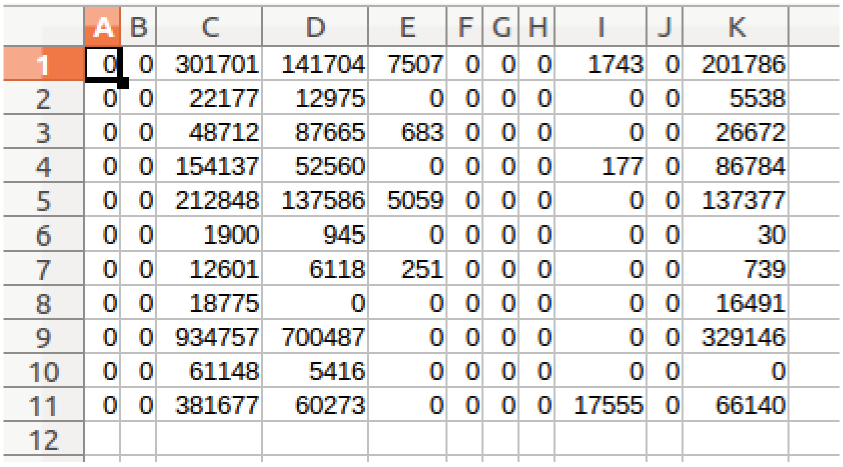
	A	B	C	D	E	F	G	H	I	J	K
1	3	1	12	4	4	0	0	8	0	0	1
2	0	4	19	15	11	0	1	6	2	1	5
3	29	55	653	137	147	1	6	312	51	0	20
4	7	34	69	294	129	1	1	49	13	1	22
5	7	19	93	126	108	0	2	70	24	2	12
6	0	0	0	0	0	0	0	0	0	0	0
7	0	1	2	2	0	0	0	1	0	1	0
8	13	27	275	87	106	0	7	247	38	4	17
9	3	4	40	10	21	0	0	37	64	0	2
10	0	0	0	1	3	0	0	0	0	1	0
11	0	5	10	14	11	0	1	9	0	0	164
12											

The corresponding conditional probabilities matrix (where the (i,j) element is the probability $v_{i|j}$ as defined above):



	A	B	C	D	E	F	G	H	I	J	K
1	0.0909090909	0.0303030303	0.3636363636	0.1212121212	0.1212121212	0	0	0.2424242424	0	0	0.0303030303
2	0	0.0625	0.296875	0.234375	0.171875	0	0.015625	0.09375	0.03125	0.015625	0.078125
3	0.0205527994	0.0389794472	0.4627923459	0.0970942594	0.1041814316	0.0007087172	0.0042523033	0.2211197732	0.0361445783	0	0.0141743444
4	0.0112903226	0.0548387097	0.1112903226	0.4741935484	0.2080645161	0.0016129032	0.0016129032	0.0790322581	0.0209677419	0.0016129032	0.035483871
5	0.0151187905	0.0410367171	0.2008639309	0.2721382289	0.2332613391	0	0.0043196544	0.151187905	0.0518358531	0.0043196544	0.0259179266
6	0	0	0	0	0	0	0	0	0	0	0
7	0	0.1428571429	0.2857142857	0.2857142857	0	0	0	0.1428571429	0	0.1428571429	0
8	0.0158343484	0.0328867235	0.3349573691	0.1059683313	0.1291108404	0	0.0085261876	0.3008526188	0.0462850183	0.0048721072	0.0207064555
9	0.0165745856	0.0220994475	0.2209944751	0.0552486188	0.1160220994	0	0	0.2044198895	0.3535911602	0	0.0110497238
10	0	0	0	0.2	0.6	0	0	0	0	0.2	0
11	0	0.023364486	0.046728972	0.0654205607	0.0514018692	0	0.0046728972	0.0420560748	0	0	0.7663551402
12											

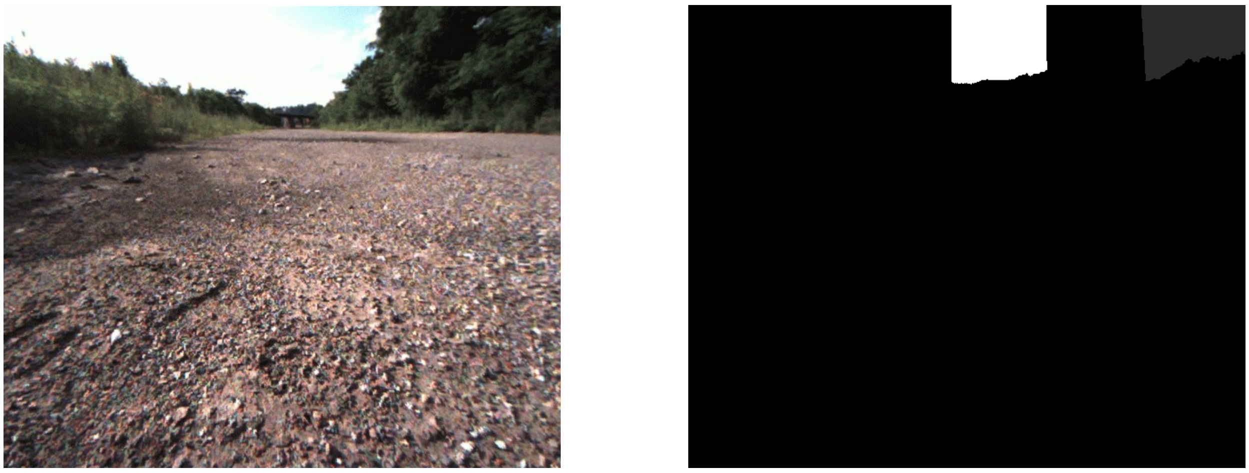
The final confusion matrix for predicted values on the 13 testing images is as shown below (available in the *'Performance_Evaluation'* folder in the *'data/Prediction_*'* directories):



	A	B	C	D	E	F	G	H	I	J	K
1	0	0	301701	141704	7507	0	0	0	1743	0	201786
2	0	0	22177	12975	0	0	0	0	0	0	5538
3	0	0	48712	87665	683	0	0	0	0	0	26672
4	0	0	154137	52560	0	0	0	0	177	0	86784
5	0	0	212848	137586	5059	0	0	0	0	0	137377
6	0	0	1900	945	0	0	0	0	0	0	30
7	0	0	12601	6118	251	0	0	0	0	0	739
8	0	0	18775	0	0	0	0	0	0	0	16491
9	0	0	934757	700487	0	0	0	0	0	0	329146
10	0	0	61148	5416	0	0	0	0	0	0	0
11	0	0	381677	60273	0	0	0	0	17555	0	66140
12											

Evidently, the only labels predicted by the classifier are 2,3,4,8 and 10 (corresponding to columns C,D,E,I and K). Clearly, there is a lot of room for improvement. Maybe using information at the edges of the superpixels more significantly can yeild better results.

This is the predicted output for one of the images (I'm pretty sure there's some glitch in the last part of the pipeline which is supposed to create this image, so hopefully I'll be uploading better predictions to the repository soon):



The actual labeling is this:

The repository doesn't contain predicted images yet. There's some debugging left to do, and the prediction/rendering process will take a while on my laptop, so I'll have 'prediction images' up by tomorrow, hopefully.

The accuracies in each fold were: 1.5%, 0.5%, 1.2% and 0.7%. I have reasons to believe that the last part of my pipeline, which is evaluating these accuracies and the prediction images has some bugs. Hopefully, when I sort them out, we'll get to seevthe real accuracies, which will be better: