HW 4

Arnab Dey
Student ID: 5563169
Email: dey00011@umn.edu

Solution 1

Summary: The dataset has total 10 attributes. One of them is sample ID number which is unique to each sample in most of the cases (though there are some duplicates). The other attributes are integer valued from 1 to 10. There are some missing values also for some samples. Therefore, the first step was to process the dataset.

Data processing: I have replaced the missing values with the mode of the feature values of the samples with no missing values. More precisely, say X_m and X_f denote the the sets of samples with any missing values and no missing values respectively. Then, for sample i with missing feature m,

$$X_m[i, m] = \text{mode}(X_f[:, m]).$$

Also, I have converted sample ID to categorical variable with two categories. The split point to divide the sample IDs into two categories are chosen to be the mean value of sample ID over all the samples. For other attributes, I tried both approaches with 10 independent categories and two categories with split point chosen as 5 (*i.e.* attribute value $\leq$ 5 denoted as 0 and 1 otherwise). I found that having two categories instead of 10 gives better generalization accuracy and reduces variance. Therefore, I have converted all the attributes to categorical with two categories with split point chosen as the mean value of the range of the values of respective categories.

Majority vote has been chosen as the predicted label for a particular leaf node. Also, in case of sample ID, it is possible that a new test sample comes with completely new sample ID which has not been seen by the tree during training, therefore, having two categorical variables helps in this case which would have not been possible if we had different branches for different values of ID.

Calculation of weighted information gain: In case of Adaboost, it is required to put different weights on samples while calculating the entropy. This can be done in two ways: (1) Calculating entropy based on weights of the samples instead of number of samples with different labels, (2) sample from training dataset with replacement according to the sample weights. I have tried both and found that, in case of approach (1), where the entropy of a particular leaf node is calculated by computing $entropy = -\frac{\sum_{i\in\mathcal{I}_p} D_i}{\sum_{i\in\mathcal{I}_p\cup\mathcal{I}_n} D_i} \log_2(\frac{\sum_{i\in\mathcal{I}_p} D_i}{\sum_{i\in\mathcal{I}_p\cup\mathcal{I}_n} D_i}) - \frac{\sum_{i\in\mathcal{I}_n} D_i}{\sum_{i\in\mathcal{I}_p\cup\mathcal{I}_n} D_i} \log_2(\frac{\sum_{i\in\mathcal{I}_n} D_i}{\sum_{i\in\mathcal{I}_p\cup\mathcal{I}_n} D_i})$, where $\mathcal{I}_p, \mathcal{I}_n$ denote the set of indices of the positive and negative samples in the leaf node and D_i denote the sample weight of sample i, it does not change if somehow Adaboost error hits 0.5 error rate. Because, once Adaboost hits 0.5 error rate, it does not change the weights of the samples in the successive iterations and therefore does not change the stumps thereafter. To avoid this, I have used sampling with replacements according to the sample weights and then fed the sampled train set to the stumps to learn. In this way we can have variations in the stump which helps in Adaboost learning.

Results: Fig. 1 shows the classification error rate as we increase the number of weak learners. It can be seen that, as the number of weak learners increases, the training error decreases. However, the test error decreases at first then freezes to decrease anymore and sometime increases also. This behavior denotes overtraining of the algorithm. Increasing the number of weak learners over-trains the algorithm and thus gives poor generalization.

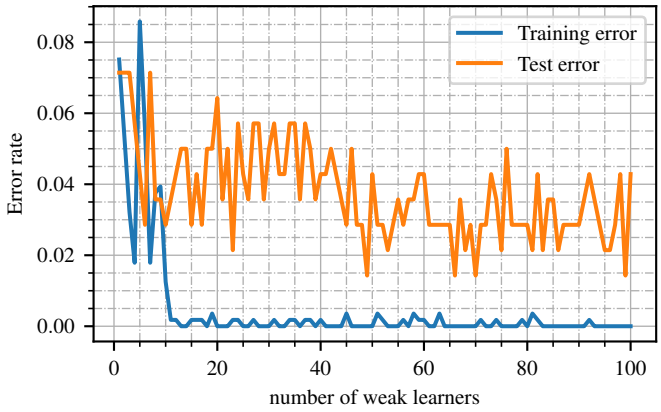


Figure 1: Q1: Adaboost: Classification error rate on train and test sets

Solution 2.i

Summary: In case of Random forest, we randomly choose m features out of $d \geq m$ features, to use for classification for each stump and learn an ensemble of B stump. In each iteration, we bootstrap a sample Z^* of size N from the training set of size N and use this data to train the stump. In this way, we train an ensemble of trees, $\{T_b\}_{b=1}^{B}$. Let, $\hat{C}_b(x)$ be the class prediction of the b^{th} tree in the random forest for the sample x. Then the prediction of the random forest for sample x is

$$\hat{C}_{rf}(x) = majority\ vote\ \{\hat{C}_b(x)\}_1^B.$$

In this homework, we are required to choose $B = 100$ and for part (i) $m = 3$.

Results: Fig. 2 shows the error rate on training and test set, with $m = 3$, as we increase the number of stumps in the random forest algorithm. It can be seen that as the number of trees increases both training and test error decreases while after a certain number of trees, the improvement of error rate on both training and test set become minimal.

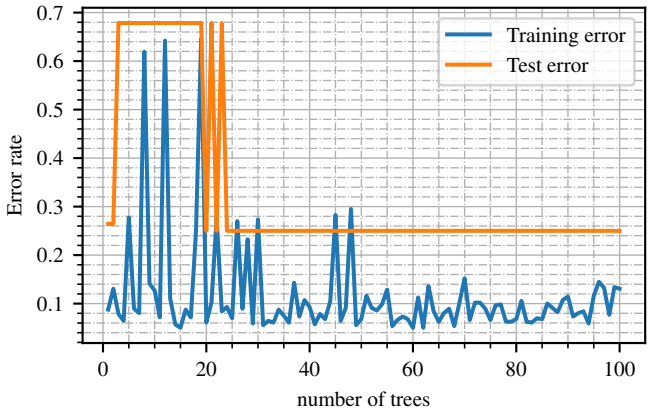


Figure 2: Q2.i: Random Forest with $m = 3$: Training and test error decreases as number of trees increases

Solution 2.ii

Summary: In this part we are required to plot the training and test error as we increase m, the number of randomly selected features to use for classification for each stump in the forest.

Results: Fig. 3 shows the plot of training and test error as we increase m. It can be seen that both training and test error decrease with increase in m, however, when m approaches d, total number of features, the

error sometimes increases due to increased variance of the tree output. As, we bootstrap the training sample set before learning, each tree is trained on different sample sets which produces poor generalization accuracy if m is equal to d. Fig. 4 shows the training error rates for different values of m as we increase the number

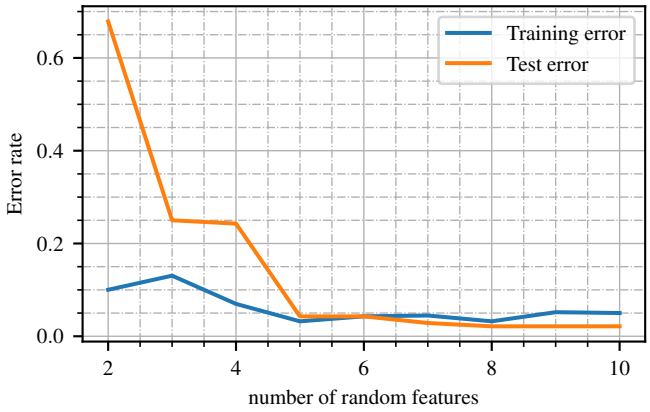


Figure 3: Q2.ii: Random Forest with varying m: Training and test error

of trees. It can be seen that increasing m, in general decreases the error rate of random forest algorithm. The figure reveals that for any particular number of trees, increasing m decreases the training error. Fig. 5

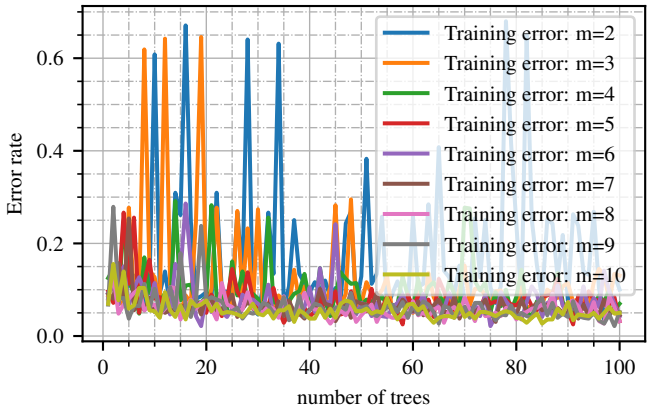


Figure 4: Q2.ii: Random Forest with varying m: Training error

shows the test error rates for different values of m as we increase the number of trees. Similar trend, as we have seen in case of training error, can be seen for test error also.

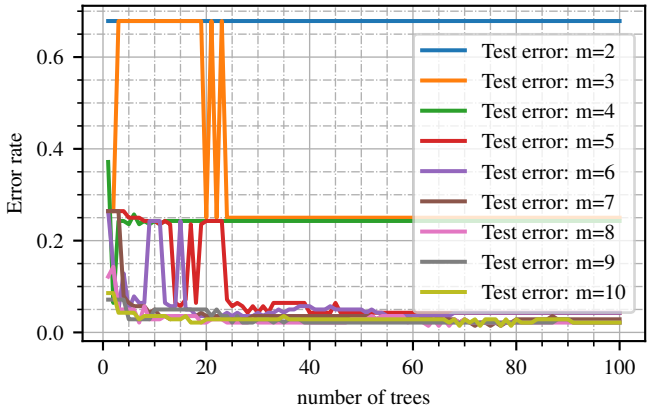


Figure 5: Q2.ii: Random Forest with varying m: Test error

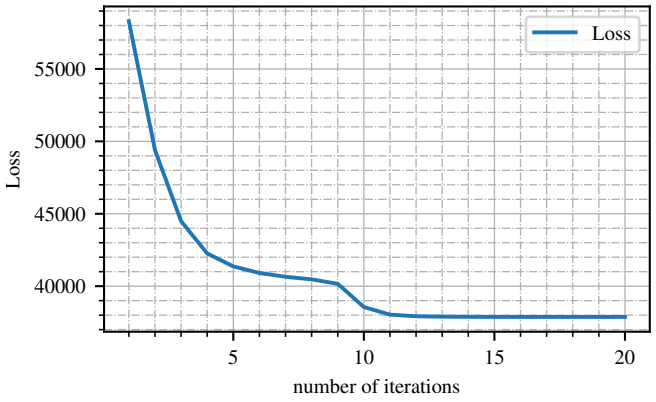
Solution 3

Summary: In the png image provided, there are total 4 channels, R, G, B and Alpha. The ranges for the values of these channels are $0 - 255$. As K-means is sensitive to the magnitudes of the vectors, I have normalized all the pixel values to $0 - 1$. At each iteration, cluster centers are initialized at random from uniform distribution over $0 - 1$. Also, I have use 10 runs for each k values and chosen the cluster means and labels of the run which produced lowest cumulative loss.

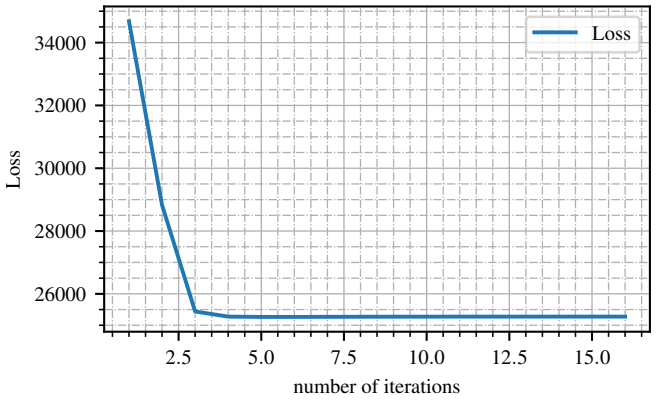
Reconstruction of the image is done as follows: If pixel i belongs to cluster C_k, then in the reconstructed compressed image, value of pixel i is replaced by the cluster mean μ_k. Thus, reduction of many different pixel values to at most k different values compresses the image.

Convergence Criteria: I have used a tolerance of $1e - 4$ in the decrease of cumulative loss between two consecutive iteration to declare convergence of the algorithm.

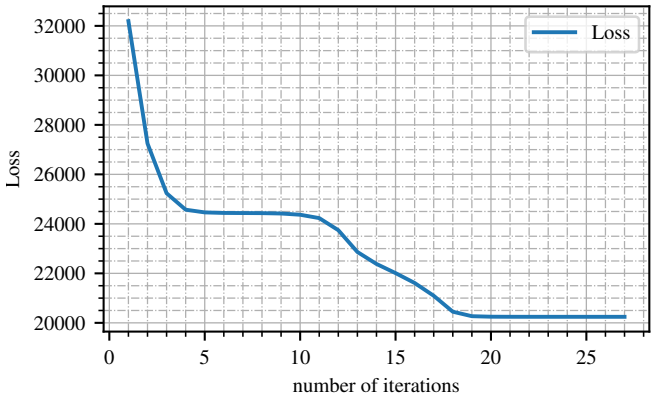
Results: Fig. 6a, 6b and 6c show the cumulative loss with iteration number in x-axis for $k = 3, 5$ and 7 respectively. It can be seen that the cumulative loss when the algorithm converges is lowest for $k = 7$ and highest for $k = 3$. It can be interpreted that $k = 7$ produces better reconstructed image at a cost of poor compression, while $k = 3$ produces better compression at a cost of poor reconstruction. Fig. 7a, 7b and 6c



(a) K-means with $k = 3$: Cumulative loss decreases with number of iterations



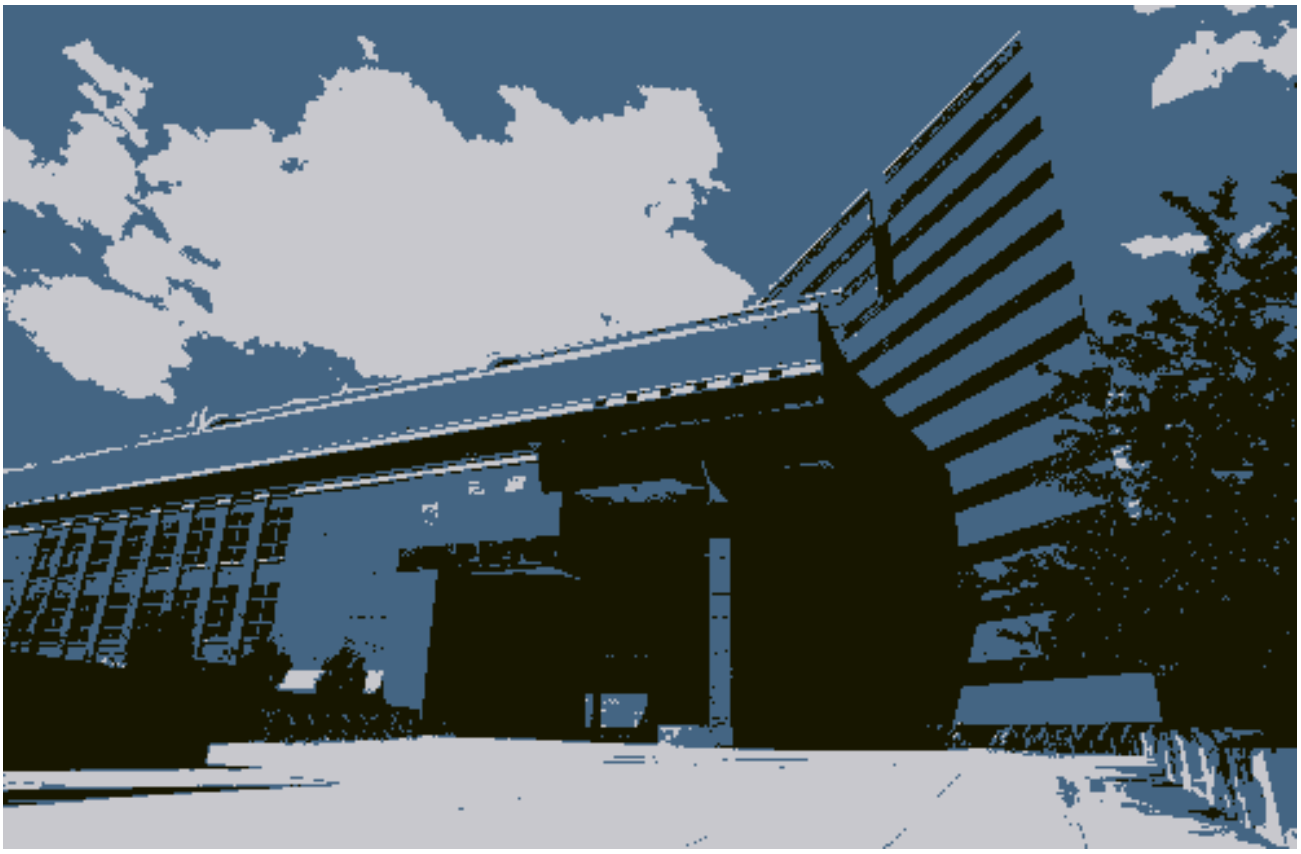
(b) K-means with $k = 5$: Cumulative loss decreases with number of iterations



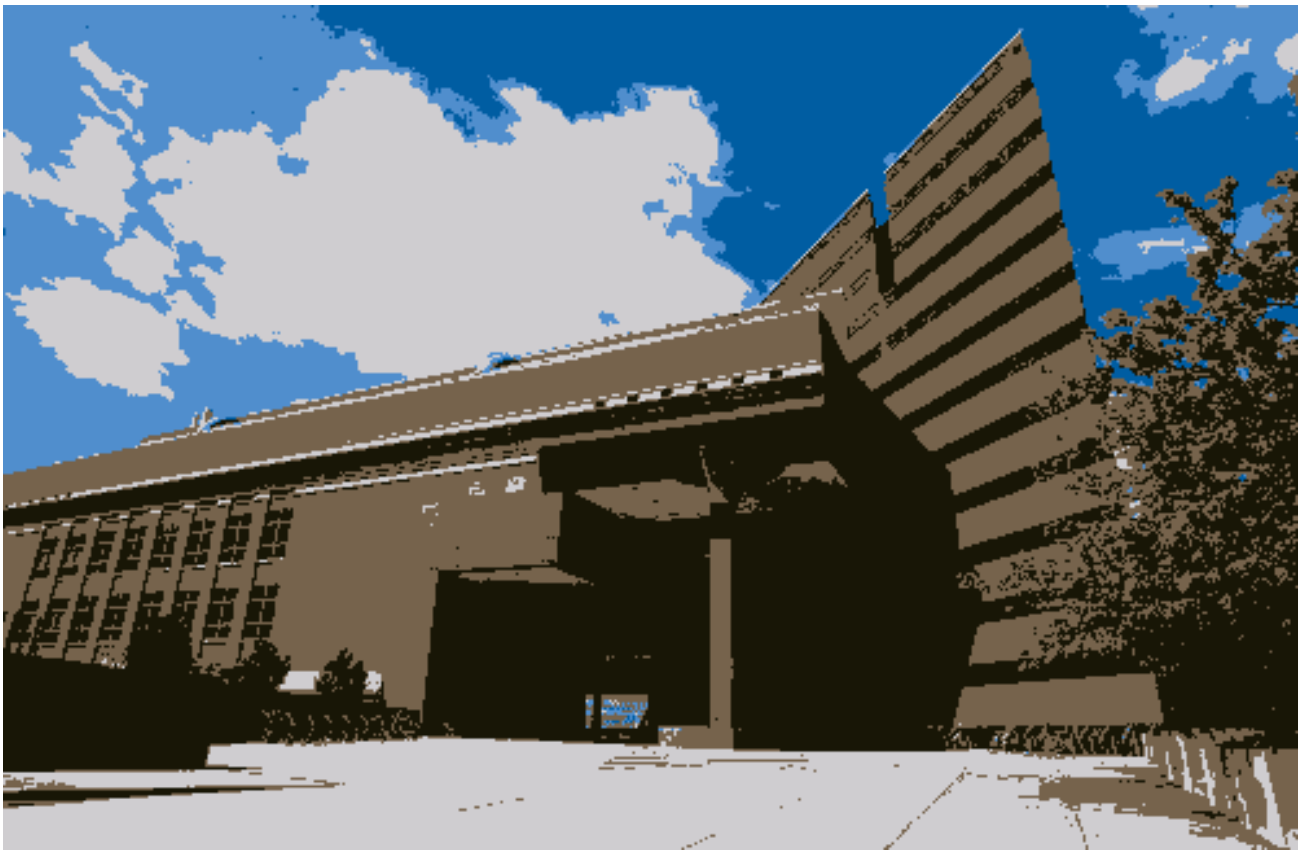
(c) K-means with $k = 7$: Cumulative loss decreases with number of iterations

Figure 6: Q3: K-means: Cumulative loss with iteration number

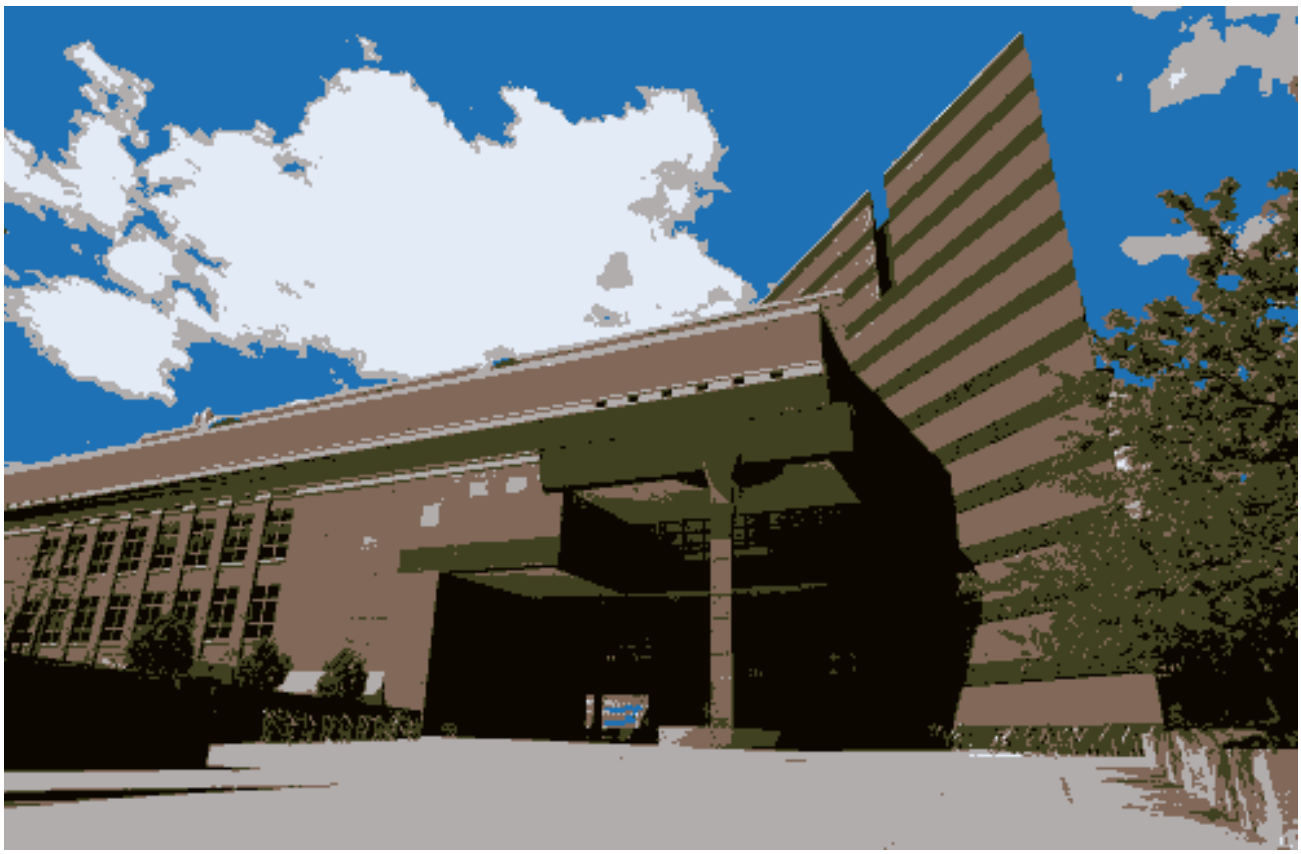
show the reconstructed image after running K-means with $k = 3, 5$ and 7 respectively. It can be seen that $k = 7$ produces the best quality image and $k = 3$ produces the lowest quality image.



(a) Q3: K-means: Reconstructed image with $k = 3$



(b) Q3: K-means: Reconstructed image with $k = 5$



(c) Q3: K-means: Reconstructed image with $k = 7$

Figure 6: Q3: K-means: Reconstructed compressed image for different k values