

520.638 Deep Learning
Homework 1 - Face Recognition

Nicholas Hinke

March 01, 2022

1 Part 1: Written Questions

1. *What do we mean by hand-crafted features? Give at least three examples of handcrafted features.*

“Handcrafted features” refer to computable properties of an image that have been explicitly defined by human researchers (which are **not** learned through training) to extract some useful discriminating information about the content of an image. Often, these features are in the form of explicit equations with well-defined inputs (*i.e.* an image) and outputs. There are many examples of handcrafted features, including Gabor features, Histogram of Oriented Gradients (HOG) features, Local Binary Pattern (LBP) features, Scale-Invariant Feature Transform (SIFT) descriptors, etc.

2. *What do we mean by learned features? Give at least three examples of learned features.*

In contrast to handcrafted features as discussed above, “learned features” are learned through training, and thus require a training dataset. Consequently, these types of features very rarely have an explicit equation definition, and are usually seen as more of a “black box”. However, the goal of learned features is the same as that of handcrafted features: to capture and represent the information within an image in a robust and useful (*i.e.* discriminating) way. There are many examples of learned features, including Visual Bag of Words (BoW), Fisher Vectors (FV), Vector of Locally Aggregated Descriptors (VLAD), etc.

3. *Briefly discuss some of the advantages and disadvantages of hand-crafted features compared to learned features.*

One of the most notable advantages of handcrafted features over learned features is that we often better understand them and where they come from. This is because of the explicit mathematical formulations of most handcrafted features, in contrast to the more “black box” nature of many learned features. Additionally, for the same reason, handcrafted features do not require a training dataset in order to be extracted from new test images. A disadvantage of handcrafted features, however, is the greater simplicity of understanding and implementation of learned features, since a more novice researcher need not understand the mathematical formulation of “black box” learned

features in order to use them. Moreover, learned features can often be incorporated directly into a classifier as a one-stage classification pipeline, whereas handcrafted features typically must first be used for feature extraction before a classifier can be applied (thus resulting in a two-stage pipeline).

4. *What are two ways in which one can formulate PCA?*

There are two equivalent ways in which one can formulate the problem of Principal Component Analysis (PCA). Given a set of samples (or feature descriptors) in high-dimensional space, the same line (*i.e.* subspace) can be found through all the samples by either: 1) maximizing the variance between the set of all samples projected onto the line or 2) minimizing the sum of the residual errors of the set of all samples projected onto the line.

5. *LDA reduces dimensionality from the original number of features to how many features?*

Due to the mathematical formulation of the Linear Discriminant Analysis (LDA) problem (specifically when solving for the optimal projection W using generalized eigenvectors), the reduced dimension of the feature space can be **at most** equal to the number of classes minus one (*e.g.* $\text{dim} = c - 1$).

6. *What are some of the limitations of LDA?*

Most notably, the reduced dimension of the feature space after performing LDA is severely restricted, and may be insufficient to capture discriminative information for many applications. For the same reason, LDA will often struggle to provide discriminative information when the number of object classes is small (due to the very low-dimensional reduced feature space). Additionally, as is the case with PCA, LDA necessarily assumes that the dataset is normally-distributed, so LDA will often perform poorly on datasets that are far from normally-distributed.

7. *Give two drawbacks of PCA.*

Since PCA necessarily assumes that the dataset follows a Gaussian distribution (like LDA), a very ‘un-Gaussian’ distribution will not be

well-described by the principal components found through PCA (*i.e.* there will be no good line through the samples or features that can maximize the variance or minimize the residual errors). Additionally, since PCA is an example of unsupervised dimension reduction, it never makes use of class labels, and therefore may not be optimal for discrimination between object classes.

8. *Many features in computer vision are represented in terms of histograms. Given two histograms, what are some distance metrics that we can use to compare them? Give at least three examples.*

There are many distance metrics that can be utilized to assess the similarity of two histograms. Three such metrics are the χ^2 distance metric, the Bhattacharyya Distance, and the Hellinger Distance.

9. *Why does the l_0 -norm capture sparsity?*

By definition, the l_0 -norm of a vector x counts the number of **non**-zero elements within x . Therefore, it is an excellent measure of the sparsity of x , since a highly sparse x with many zeros will have a very low l_0 -norm (*i.e.* low l_0 -norm \Leftrightarrow high sparsity). In other words, minimizing the l_0 -norm of a vector is essentially equivalent to maximizing the sparsity of that vector.

10. *Why do we use the l_1 -norm to approximate the l_0 -norm?*

Since the l_0 -norm optimization problem is a **non**-convex problem, it is extremely challenging (and computationally intensive!) to even attempt to solve. Consequently, we use the l_1 -norm to approximate it, since the l_1 -norm yields a nice convex optimization problem for which well-defined methods exist to solve. Additionally, under certain conditions related to restricted isometries, the solution to the l_1 -norm problem is equivalent to that of the l_0 -norm problem (at least as it relates to sparse coding).

11. *What are some disadvantages of k -means clustering?*

Most notably, k -means clustering considers only the centroid of each cluster (known as “hard assignment”), and therefore ignores potentially useful information regarding the covariance structure of each cluster. Additionally, especially since k -means clustering is an unsu-

pervised process, it is extremely important to have a sufficiently representative training set of everything that could be seen during testing; otherwise, the information extracted by the clusters (known as the “codebook” when considering BoW features) will **not** be sufficiently universal.

12. *What is the difference between Nearest Neighbor algorithm and k-Nearest Neighbor algorithm?*

In the Nearest Neighbor (NN) algorithm, a new test image is assigned the same label as that of the single most similar image in the entire training set (which is found using a given distance metric such as the Euclidean distance). By contrast in the k-Nearest Neighbor (k-NN) algorithm, a new test image is assigned the label of the majority vote from all ‘k’ nearest neighbors (again, found using a given distance metric). The value of ‘k’ can be considered as a tunable hyperparameter, and typically leads to superior performance for k-NN over NN. It should also be noted that NN and k-NN are exactly equivalent when the value of ‘k’ is one.

13. *Briefly describe how visual bag of words (BoW) features are extracted.*

As Visual BoW features were adopted from the natural language processing (NLP) community, the motivation behind them is as follows: to represent images as a collection of salient features like you could represent a book as a collection of words (or how you could represent a spoken word as a collection of sounds). The extraction of BoW features is typically done in four stages: 1) use some method to extract features from an image, 2) learn a redundant “visual vocabulary” that can represent any image using the features from stage one, 3) quantize features from the visual vocabulary to capture salient information and reduce redundancy (*e.g.* use k-means clustering where each cluster centroid represents a “visual word”), and 4) represent the image using a histogram of the frequencies of the various visual words in the visual vocabulary.

14. *Briefly describe cross-validation.*

The goal of cross-validation (CV) is to find the set of hyperparameters that maximizes the performance of a given model. This is done by splitting only the **training** dataset into a smaller training set and

a validation set, where the validation set acts as a fake test set for tuning the hyperparameters. We can then select the hyperparameters that perform the best on the validation set as “optimal”, and then report the model’s performance on the **test** set using those parameters. In practice, this is usually done by splitting the training data into small portions (or “folds”), where each fold is treated as the validation set and then all of the results are averaged (known as “k-Fold CV”).

15. *What is the difference between sparse coding and dictionary learning?*

Sparse coding and dictionary learning both rely on the same underlying motivation, which is that natural images can be effectively represented using the linear combination of a few atomic “dictionary” elements (*e.g.* surfaces or edges). The key difference between the two techniques is that sparse coding requires that the dictionary (*i.e.* a set of redundant basis elements) be known *a priori*, whereas dictionary learning simultaneously incorporates the generation (or “learning”) of the dictionary along with the computation of the sparse representations.

2 Part 2: Face Recognition using k-NN Algorithm

For this section, the k-Nearest Neighbor (k-NN) algorithm was implemented and evaluated using a variety of different parameters for face recognition using the Extended YaleB dataset (note that all images from this dataset were cropped to a size of 32x32). The dataset contains 38 object classes (unique individuals), for each of which there are approximately 64 frontal images. It should also be noted that throughout this section, all code used to complete these problems was written in Python (and is publicly available at <https://github.com/nhinke/deeplearning-repo/tree/master/hw1/src>) with the use of the following libraries:

- `numpy`
- `scipy.io`
- `matplotlib.pyplot`
- `skimage.feature.hog`
- `skimage.feature.local_binary_pattern`
- `sklearn.neighbors.KNeighborsClassifier`

2.1 Extended YaleB Dataset

1. Using the YaleB dataset, ($m = 10, 20, 30, 40, 50$) images were randomly selected from each individual class (with labels) to form the training set, and the remaining images were used as the test set. The k-NN algorithm (with $k = 1, 2, 3, 5, 10$) was then applied on each of these five splits (values of m), and the corresponding classification errors were recorded. Note that the Euclidean distance metric (*i.e.* $d(x, y) = \|x - y\|_2$) was used. The classification error rate was then plotted against the number of training samples per class (m) on a single figure. Note that all code used for this problem can be found publicly at <https://github.com/nhinke/deeplearning-repo/tree/master/hw1/src>. The resulting plots were as follows:

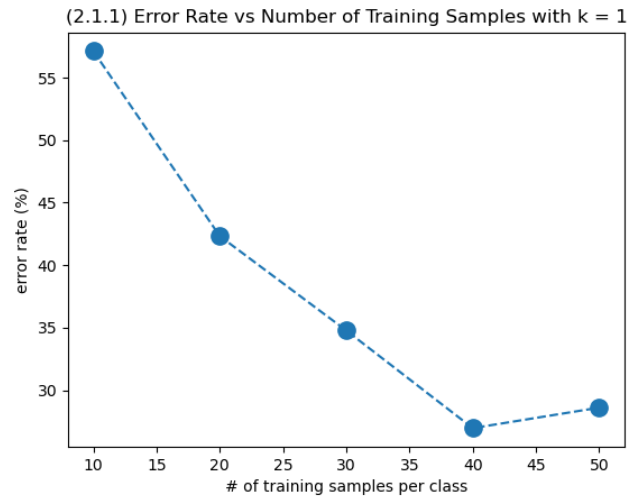


Figure 1: Problem (2.1.1) Classification Error Rates with $k = 1$

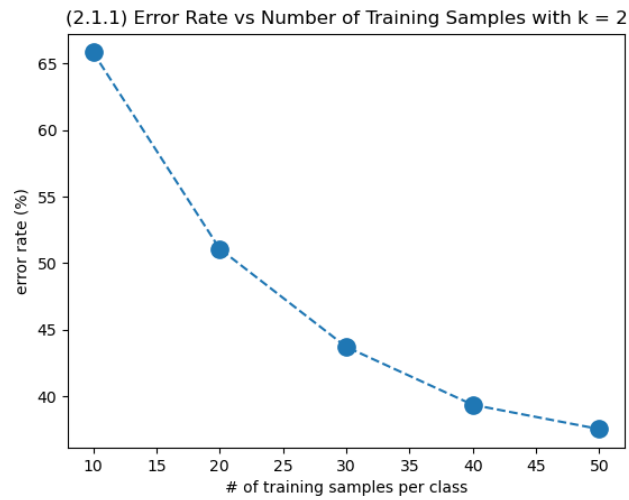


Figure 2: Problem (2.1.1) Classification Error Rates with $k = 2$

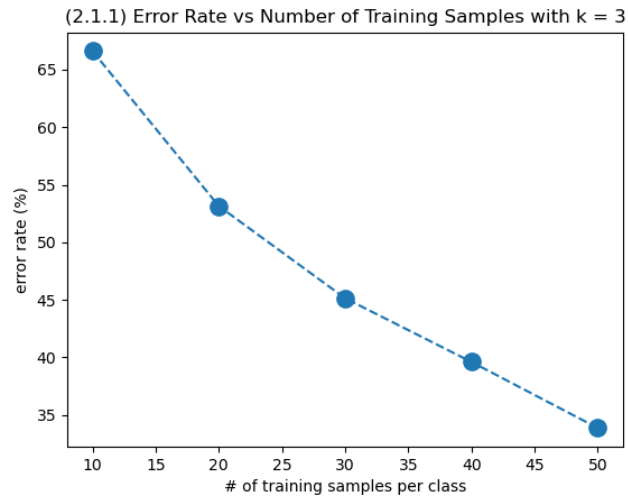


Figure 3: Problem (2.1.1) Classification Error Rates with $k = 3$

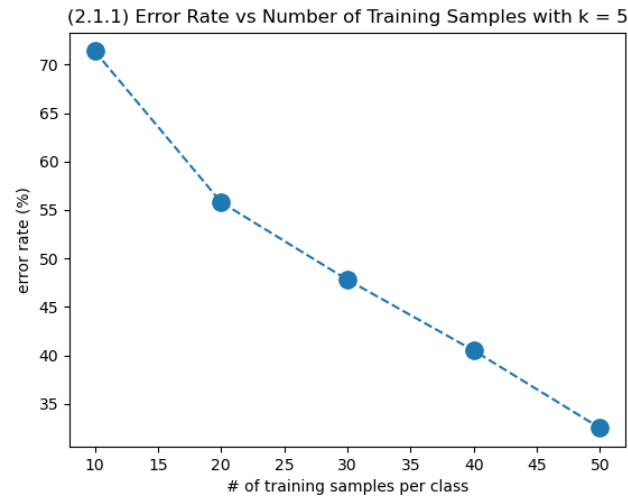


Figure 4: Problem (2.1.1) Classification Error Rates with $k = 5$

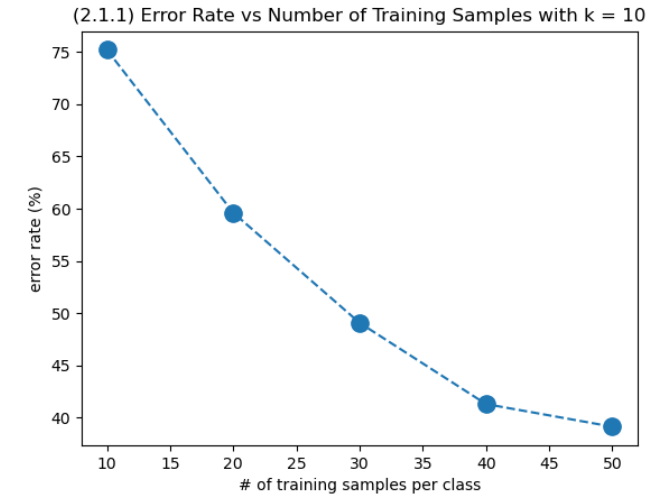


Figure 5: Problem (2.1.1) Classification Error Rates with $k = 10$

As evidenced by the performance in the plots above, the k -NN typically performs better for greater values of m , but does not always. This may be the result of a case of overfitting to the training data.

2. The same experiment was performed as in problem (2.1.1), but this time the classification error rates were plotted against the number of neighbors k . Note that all code used for this problem can be found publicly at <https://github.com/nhinke/deeplearning-repo/tree/master/hw1/src>. The resulting plots were as follows:

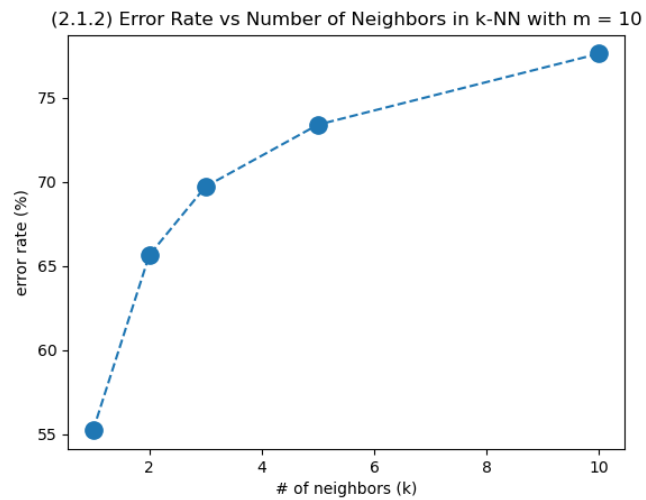


Figure 6: Problem (2.1.2) Classification Error Rates with $m = 10$

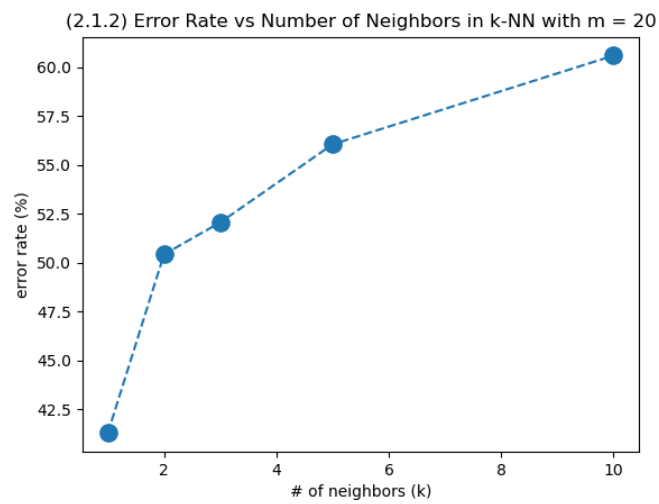


Figure 7: Problem (2.1.2) Classification Error Rates with $m = 20$

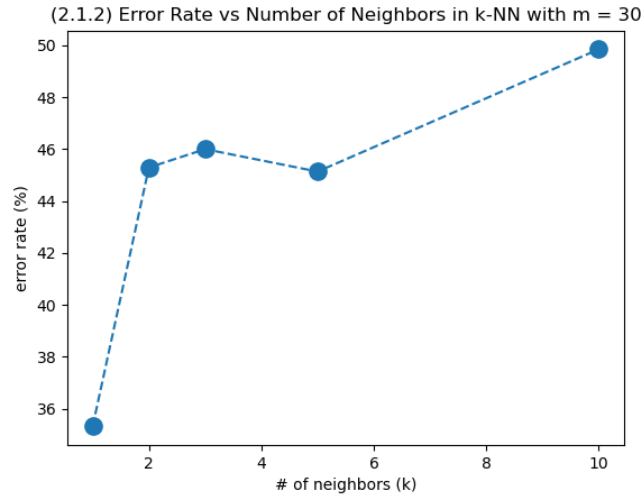


Figure 8: Problem (2.1.2) Classification Error Rates with $m = 30$

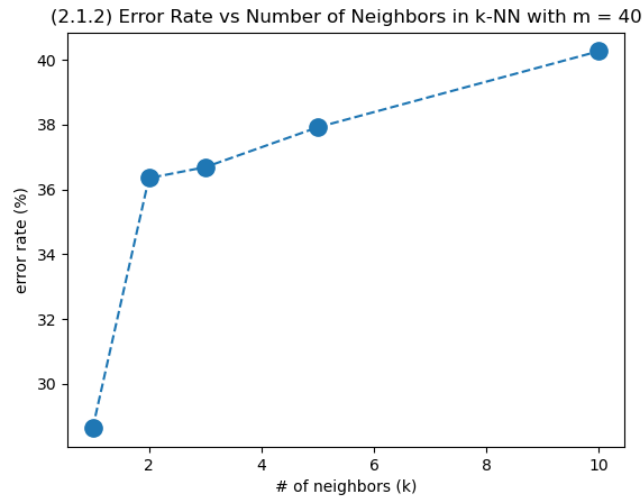


Figure 9: Problem (2.1.2) Classification Error Rates with $m = 40$

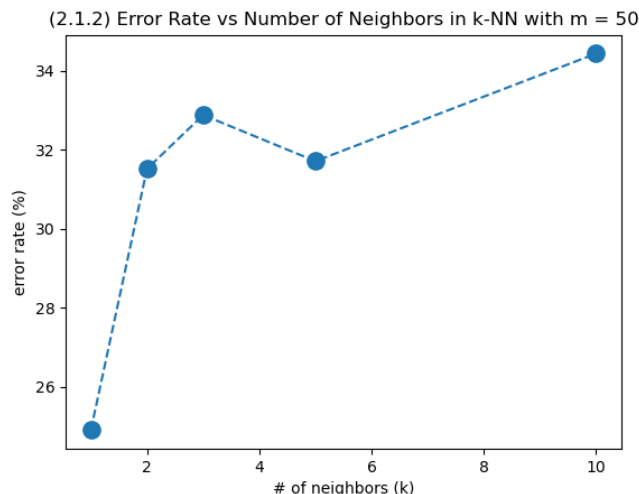


Figure 10: Problem (2.1.2) Classification Error Rates with $m = 50$

Clearly, as seen from the plots above, the classification error rate does **not** always decrease as the number of neighbors k increases; in fact, the k-NN classifier typically performs best when $k = 1$ in these experiments. Indeed, the classification error rate should not always decrease as k increases, due to the fact that the model complexity decreases beyond its optimal point as k continues to increase.

3. A similar experiment to those above was completed by letting $k = 3$ and $m = 30$, and by varying the p -norm used in the distance metric with $p = 1, 2, 3, 5, 10$ (*i.e.* $d(x, y) = ||x - y||_p$). The classification error rates were then plotted against the utilized p values. Note that all code used for this problem can be found publicly at <https://github.com/nhinke/deeplearning-repo/tree/master/hw1/src>. The resulting plot was as follows:

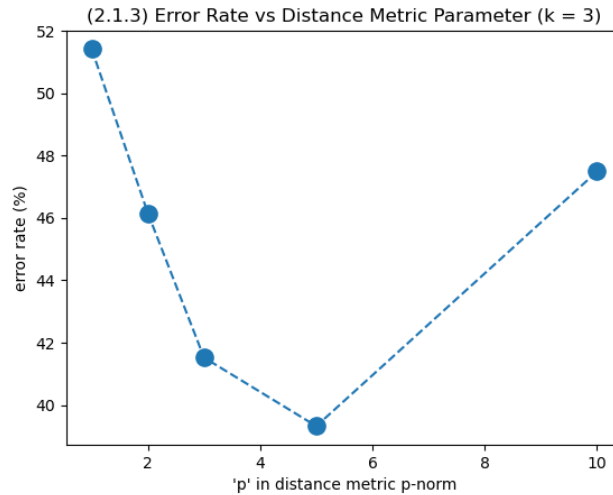


Figure 11: Problem (2.1.3) Classification Error Rates with $k = 3$

As can be seen from the plot above, the p value in the distance metric does **indeed** affect the classification error rate, allowing the classifier to achieve its best performance when $p = 5$.

4. The same experiment as described above in problem (2.1.3) ($k = 3$ and $m = 30$) was then repeated for $p = 1, 2$ using other feature descriptors rather than just raw pixel intensities. Namely, HOG features and LBP features were also tested. The corresponding classification error rates were then plotted against the utilized p values. Note that all code used for this problem can be found publicly at <https://github.com/nhinke/deeplearning-repo/tree/master/hw1/src>. The resulting plots were as follows:

(2.1.4) Error Rate vs Distance Metric Parameter using Feature = Pixel Intensities ($k = 3$)

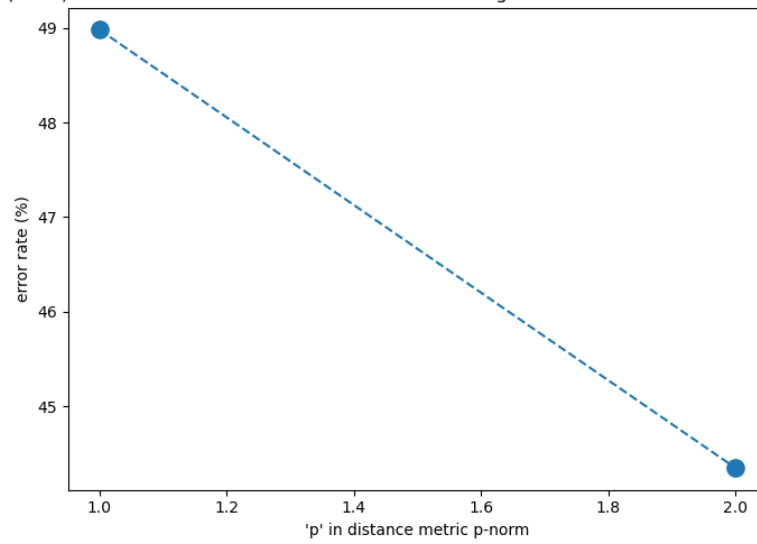


Figure 12: Problem (2.1.4) Classification Error Rates using Pixel Intensities

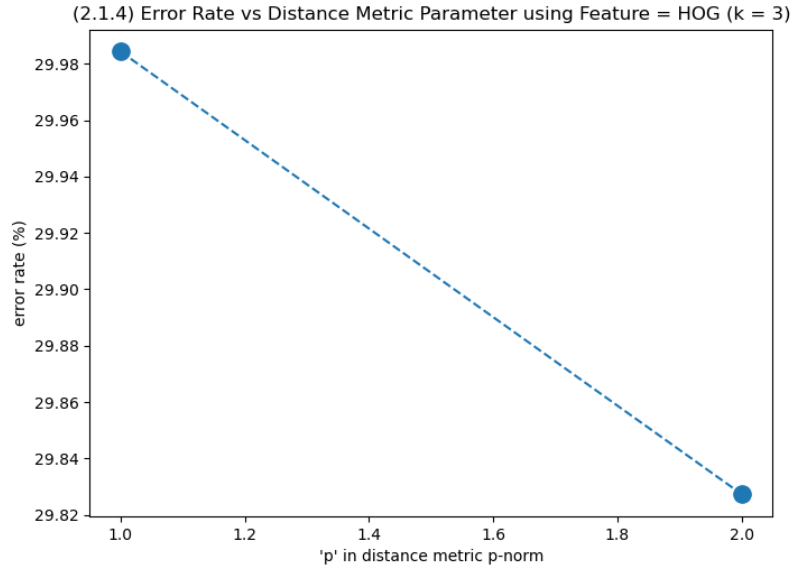


Figure 13: Problem (2.1.4) Classification Error Rates using HOG features

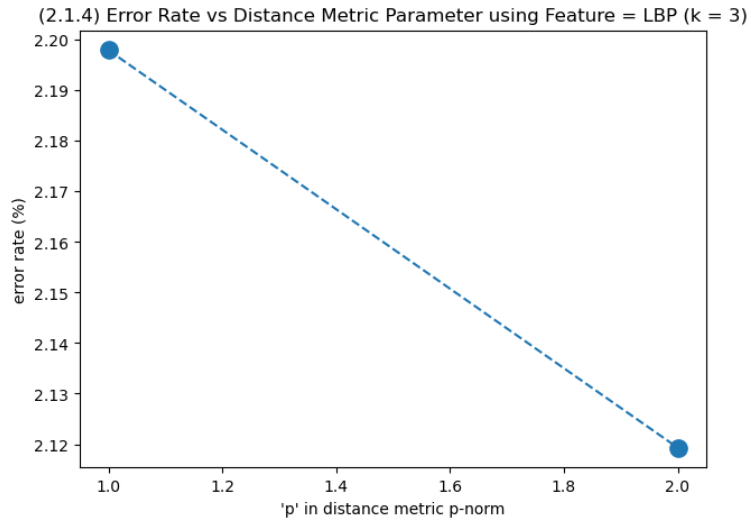


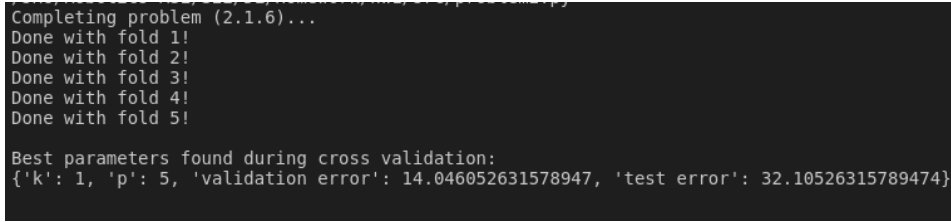
Figure 14: Problem (2.1.4) Classification Error Rates using LBP features

As can be clearly seen in the plots above, the k-NN performed the best **by far** when using LBP feature descriptors for training and testing (in comparison to HOG features or raw pixel intensities).

5. As mentioned briefly above in problem (2.1.4), the k-NN classifier performed the best when using LBP (local binary pattern) feature descriptors. This particular experiment was performed with $k = 3$ and $m = 30$, and a number of trials of this experiment demonstrated that the difference in performance was essentially negligible when choosing a p value of 1 or 2 within the distance metric. More specifically, the k-NN classifier using these parameters with LBP features achieved a classification error rate of just over 2%.

2.2 Validation Set

After performing the experiments outlined above, new experiments were performed involving cross-validation. First, 20 images per image class (*i.e.* per individual) were randomly selected to form a test set, and the remaining data was kept as the training set. Then, only the **training** set was further divided into a new smaller training set and a validation set. Cross-validation was then performed using k -Fold CV (with $k = 5$) in order to optimize the hyperparameters k and p within the k-NN classifier. It should be noted that raw pixel intensities were used as the feature descriptors for all experiments in this problem. Note that all code used for this problem can be found publicly at <https://github.com/nhinke/deeplearning-repo/tree/master/hw1/src>. The results were then printed to the active terminal as follows:



```
Completing problem (2.1.6)...  
Done with fold 1!  
Done with fold 2!  
Done with fold 3!  
Done with fold 4!  
Done with fold 5!  
  
Best parameters found during cross validation:  
{'k': 1, 'p': 5, 'validation error': 14.046052631578947, 'test error': 32.10526315789474}
```

Figure 15: Problem (2.2) Resulting Hyperparameters found using CV

As can be seen in the screenshot above, the hyperparameters $k = 1$ and $p = 5$ were found to perform best after testing all combinations of $k = 1, 2, 3, 5, 10$ and $p = 1, 2, 3, 5$. Note that the resulting validation error and test error using the optimal parameter set are also reported in the screenshot above.

3 Part 3: Face Recognition using Other Algorithms

After performing the variety of experiments using a k-NN classifier as outlined above, a variety of other classification algorithms for face recognition on the YaleB dataset were also implemented. Namely, these included: Eigenfaces (PCA), Fisherfaces (LDA), Support Vector Machine (SVM), and Sparse Representation-based Classification (SRC). The same procedure as outlined in problem (2.1.1) was followed using ($m = 10, 20, 30, 40, 50$) randomly selected images per object class (*i.e.* per individual) as the training set. Additionally, as was done previously, the Euclidean distance error metric (*i.e.* $d(x, y) = \|x - y\|_2$) was once again utilized. It should also be noted that throughout this section, all code used to complete these problems was written in Python (and is publicly available at <https://github.com/nhinke/deeplearning-repo/tree/master/hw1/src>) with the use of the following libraries:

- `numpy`
- `scipy.io`
- `matplotlib.pyplot`
- `sklearn.model_selection.GridSearchCV`
- `sklearn.svm.SVC`
- `sklearn.decomposition.PCA`
- `sklearn.decomposition.SparseCoder`
- `sklearn.discriminant_analysis.LinearDiscriminantAnalysis`

For these four experiments, the classification error rates were once again plotted against the number of training samples per class (m). Note that all code used for this problem can be found publicly at <https://github.com/nhinke/deeplearning-repo/tree/master/hw1/src>. The resulting plots were as follows:

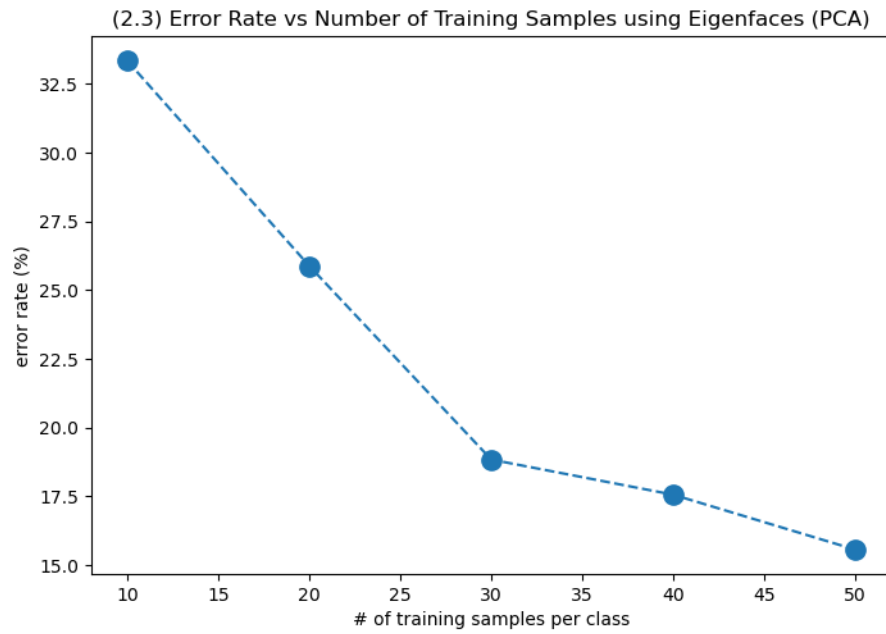


Figure 16: Performance of Eigenfaces (PCA) classifier on YaleB dataset

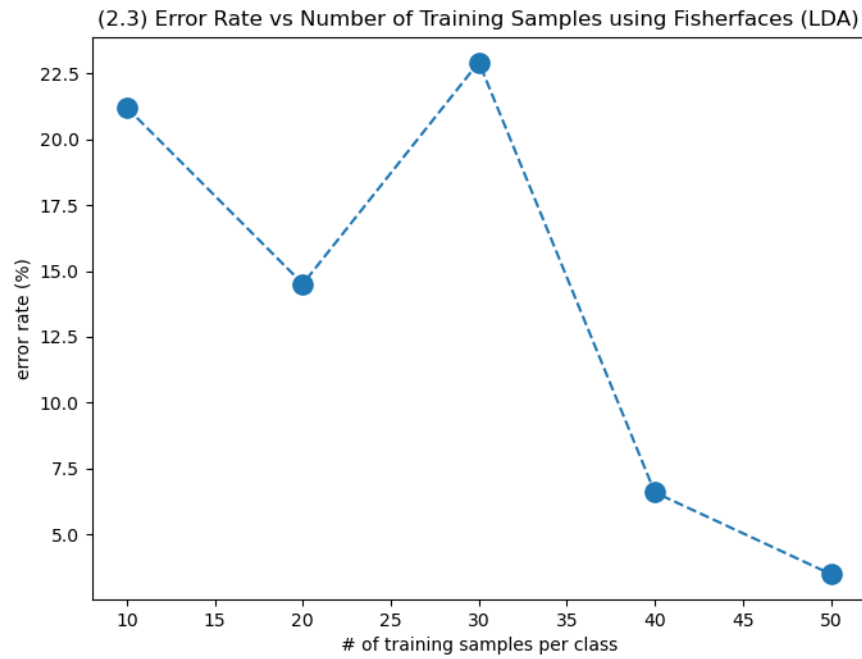


Figure 17: Performance of Fisherfaces (LDA) classifier on YaleB dataset

(2.3) Error Rate vs Number of Training Samples using Support Vector Machine (SVM)

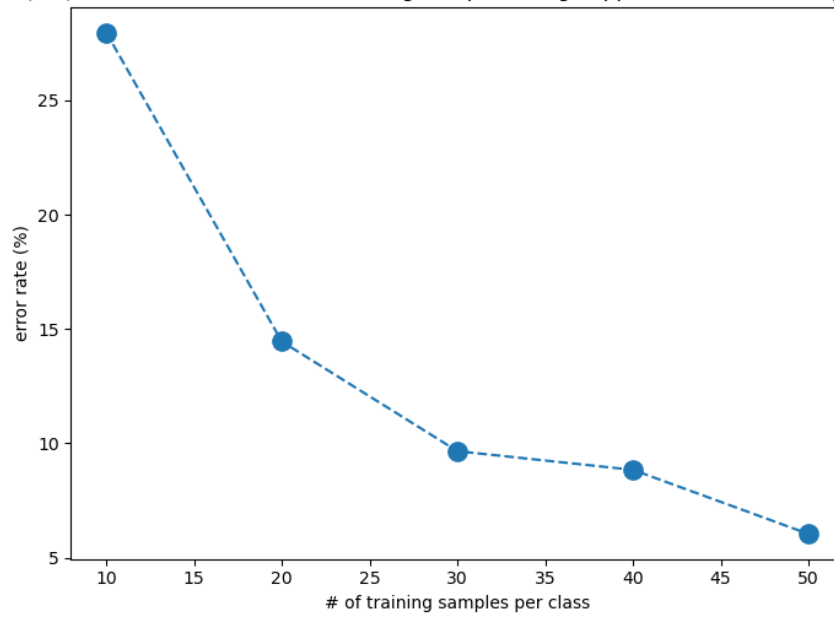


Figure 18: Performance of SVM classifier on YaleB dataset

(2.3) Error Rate vs Number of Training Samples using Sparse Representation-based Classification (SRC)

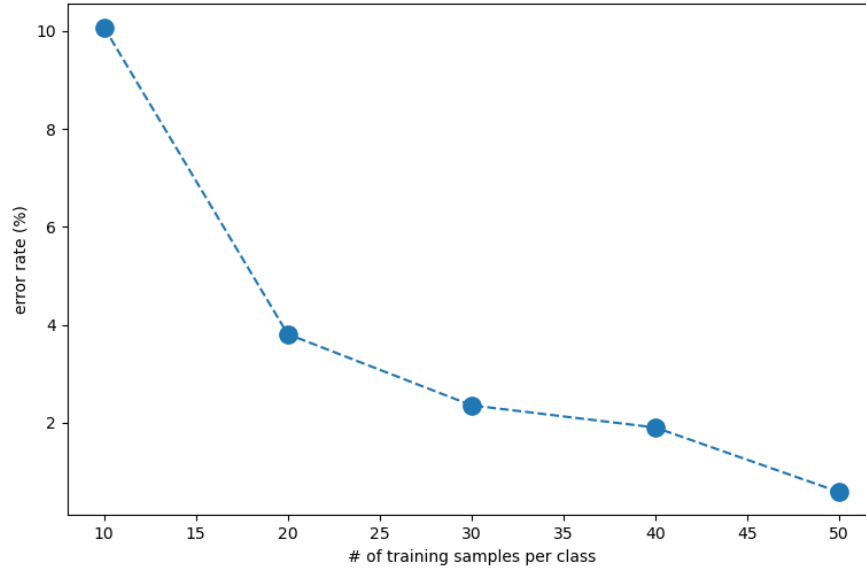


Figure 19: Performance of SRC classifier on YaleB dataset

As can be seen from the plots above, the SRC classifier vastly outperforms any of the other classifiers, reaching its best performance at an error rate of only approximately 0.75%. This corroborates some of the claims made by the original authors regarding the superiority of the algorithm when it was initially developed. It is also worth noting that the Fisherfaces algorithm does indeed outperform the Eigenfaces algorithm as was discussed in class. Finally, the SVM classifier also performs quite well, and it is *very* clear from this set of experiments that these four algorithms from problem 3 are capable of substantially outperforming the simpler k-NN classifiers as implemented in problem 2 (note that this has only been confirmed with certainty when using raw pixel intensities as features). That being said, however, the k-NN classifiers from problem 2 still perform well and are quite useful, especially when considering their simplicity and ease of implementation.