# CS492(A) Machine Learning for Computer Vision

Anar Rzayev
KAIST
rzayev.anar1@kaist.ac.kr

Berhane Weldegebriel
KAIST
breteklu@kaist.ac.kr

## Abstract

*In this coursework, we will present an efficient approach to face recognition, using the Principle Component Analysis for dimensionality reduction and multi-class classifiers, such as Nearest Neighbours. Extensive research is conducted to find the parameters that maximise recognition accuracy. Moreover, we will tackle different methods for evaluating performance of the Random Forest classifier on* ***Caltech 101*** *dataset, using k-Means clustering and RF embeddings.*

## 1. Experiment with face dataset

### 1.1. Dataset

The dataset used in this experiment contains $520$ images of size $46 \times 56$. The dataset grouped into $52$ classes each containing $10$ images. We split the dataset into training and testing data in $8:2$ ratio leading to cardinality $N = 416$ and $N = 104$. To ensure unbiased class representation, we utilized the technique of stratified sampling to apply class-wise split. The training data was normalized by subtracting the mean face from each image as: $X_{\text{normalized}} = \frac{1}{N}\sum_{i=1}^{N} x_i$ where $N$ is the number of data instances and $X_i$ is $i$th image vector. No further augmentations or transformations were done on the training data.

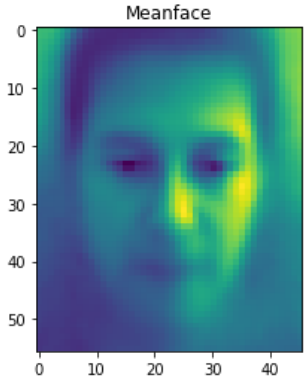



Figure 1. Mean face of our dataset

### 1.2. Principal Component Analysis (PCA)

PCA is a widely used technique for reducing dimensionality of data. In our dataset, the number of data instances $N = 520$ is much smaller than the number of features, $D = 2576$ per each instance. By applying PCA on our data, we aim to project our data from $2576$ dimensional higher space to a lower dimension $M$ where $M \leq D$. We intend to capture the most important information from the face dataset so that the data can be compressed for improved spatial and computational complexity, while maintaining the capability to be reconstructed back to the original $2576$ dimensional space.

#### 1.2.1 Naive PCA

Naive PCA finds a new basis for the given data from the eigenvectors of the covariance matrix $S_{naive} = \frac{1}{N}AA^T$, where $A$ is a matrix of normalized training face dataset. $S_{naive}$ is a real and symmetric matrix and $A$ is a matrix with dimensions $R^{D \times N_{train}}$. From basic linear algebra, $rank_A \leq \min(D, N_{train})$ and $Rank(A) = Rank\left(AA^T\right) = Rank\left(S_n\right)$. This means that $rank_{S_n} \leq N_{train}$ and $S_n$ will have $D - N_{train}$ real non-zero eigenvalues at most. Calculating the eigenvalues of $S_n$, we get $N_{train} - 1$ non zero eigenvalues. These eigenvalues are real as predicted from theoretical analysis. The problem with this approach is that it operates on a $2576 \times 2576$ matrix so it is not time and memory efficient. This leads to the use of an efficient PCA algorithm.

#### 1.2.2 Efficient PCA

Efficient PCA takes advantage of the symmetry of $S_n$ to perform eigen decomposition on $S_{efficient} = \frac{1}{N}A^TA \in R^{N_{\text{train}} \times N_{\text{train}}}$. This approach is more practical as it leads to ideantical results with $N_{train} << D$, in this case $S_{efficient}$ matrix is of size $416 \times 416$ as opposed to $2576 \times 2576$.

$$\mathcal{S}_e\mathbf{v} = \lambda\mathbf{v} \iff \frac{1}{N}A^TA\mathbf{v} = \lambda\mathbf{v}$$

| M | Accuracy |
|---|---|
| 10 | 0.6624 |
| 50 | 0.8746 |
| 100 | 0.9358 |
| 200 | 0.9778 |

Table 1. Reconstruction accuracy

Multiplying both sides with $A$ :

$$\frac{1}{N}AA^TA\mathbf{v} = \lambda A\mathbf{v} \Longleftrightarrow \mathcal{S}_nA\mathbf{v} = \lambda A\mathbf{v}$$

Let $\mathbf{u} = A\mathbf{v}$ :

$$\mathcal{S}_n\mathbf{u} = \lambda\mathbf{u}$$

So, $\mathcal{S}_{naive}$ and $\mathcal{S}_{efficient}$ have the same eigenvalues $\lambda$ and in addition, their eigenvectors follow the relation:

$$\mathbf{u}_{\mathcal{S}_n} = A\mathbf{v}_{\mathcal{S}_e}$$

To prove the theoretical computations with experiment, we ran this technique on our data and saw that the number of non-zero eigenvectors was $N_{train} - 1$. To further check the equivalence of the eigenvalues obtained from the two methods, we summed the absolute difference between the eigenvectors leading to a zero vector. Figure 15 in the appendix shows the equality of the eigenvalues.

#### 1.2.3 Comparison of the two methods

Since the dataset at hand has a nature of $N_{train} << D$, we noticed a considerable difference in performance of the two methods. Naive approach took around $14.23$ seconds to compute while efficient approach took $0.29$ seconds. In doing naive PCA, since the number of independent eigenfaces is limited to $N_{train}$ , storing an additional $D - N_{train}$ eigenfaces is not necessary. In that regard, we will use efficient PCA for the next sections.

### 1.3. PCA applications in face image reconstruction

Reconstruction of our $M$ dimensional feature space embedding of the provided dataset (where $M < D$) can be done by using the projection matrix $U$ composed of eigenfaces. An image can be reconstructed as $x_{reconstructed} = x_{mean} + Uw_n$, where $x_{mean}$ is the mean face of our dataset and $U$ is an $D \times M$ projection matrix made up of $M$ eigenfaces.

Thus as we increase the dimensionality of the embedding space (increase the number of eigenfaces), we get better reconstruction outputs as described in Table 1. An example of such reconstruction is shown in Figure 2. More examples can be found in the appendix.

## 2. Incremental PCA

Incremental principal component analysis is typically used as a replacement for principal component analysis when the dataset to be decomposed is too large to fit in memory. Incremental PCA builds a low-rank approximation for the input data using an amount of memory which is independent of the number of input data samples. It is still dependent on the input data features, but changing the batch size allows for control of memory usage.

For illustrating how the existing eigenspace model updates using incremental PCA, we have to first divide training dataset equally into $4$ subsets, every of which contains $104$ images.

| | Time(sec) | Memory(KiB) | Accuracy |
|---|---|---|---|
| Incremental | 1.3 | 129000 | 0.529 |
| Batch | 0.56 | 6970 | 0.686 |
| PCA trained by one subset | 0.8 | 1600 | 0.325 |

Table 2. Comparison for 3 different techniques

### 2.1. Performance and Parameters

One of the important parameters influencing incremental PCA is the number of significant eigenvectors to choose after merging procedure. It is a well-known fact that if $d_1$ and $d_2$ denote the top significant eigenvectors from eigenspace models, then the desired number of eigenvectors will become $d_1 + d_2 + 1$ while combining those $2$ eigenspace models. According to Table 2, keeping top $100$ eigenvalues during incremental PCA will result in $52.9\%$ accuracy after combining smaller groups. However, it should be mentioned how incremental PCA requires considerable memory space and time complexity due to the reconstruction of covariance matrices and orthonormalization processes.

In fact, if we assume $N << D$, one has to set the number of eigenvectors $d_1$ and $d_2$ considerably smaller than those

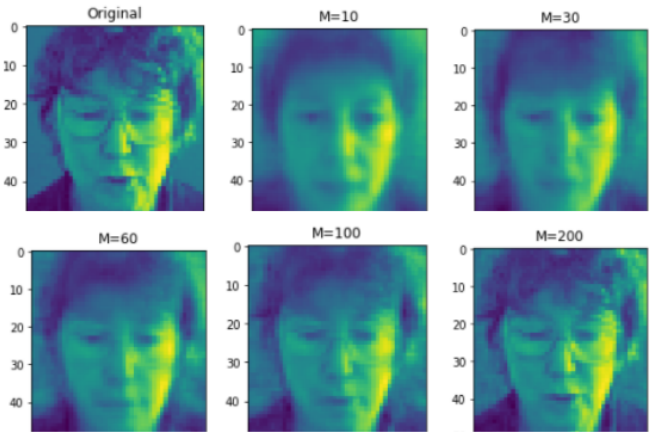



Figure 2. Reconstruction for various principal components used

for $N_1$ and $N_2$ for depicting *incremental PCA* vs *batch PCA* graph during the training period. Taking into account trade-off between accuracy and efficiency, we have to manage $d_3$ being almost a constant value, in which one has to remove less significant eigenvectors after merging process of $d_1$ and $d_2$, while maintaining the significant $d_3$ components.

### 2.2. Reconstruction Error

Using the previously applied image from problem 1, the following graph illustrates reconstruction error up to parameter $M$ being equal to 200. The decreasing error results are observed from the Figure 3.

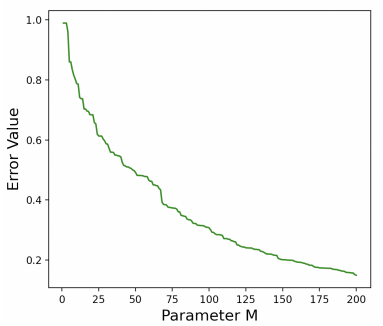



Figure 3. Incremental PCA Reconstruction Graph

## 3. PCA-LDA for face recognition

In search of a model that has the best accuracy, there are two parameters which can be tuned: $M_{PCA}$, the amount of PCA vectors used and $M_{LDA}$, the amount of LDA vectors used.

Figure 4 shows the relationship between $M_{PCA}, M_{LDA}$ and error of the model. Increasing the number of LDA vectors beyond 50 leads to decreased accuracy as overfitting occurs. Increasing $M_{LDA}$ beyond a certain point also leads to decreased accuracy. Certain combinations of $M_{PCA}$ and $M_{LDA}$ give better accuracies and by experimenting, the best accuracy we got is $91.29\%$ when $M_{PCA} = 146$ and $M_{LDA} = 51$.

## 4. Optional: PCA-LDA Ensebmle (Theoretical Analysis)

Ensemble learning improves the performance of PCA-LDA-NN classifier. There are different techniques to do this like either using random data samples or using random model parameters. By combining classifiers, Ensemble learning technique improves on performance. However, there needs to be a low amount of correlation between the models for this technique to give better accuracy.

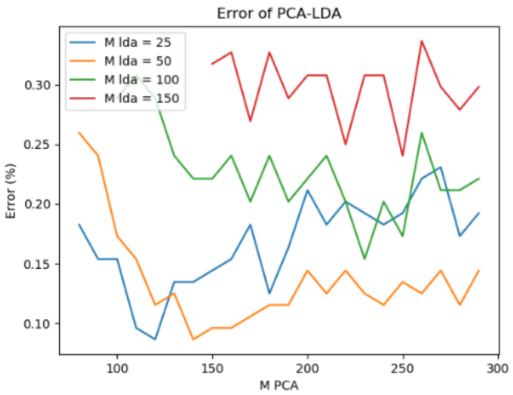



Figure 4. Classification error with different $M_{PCA}$ $M_{LDA}$

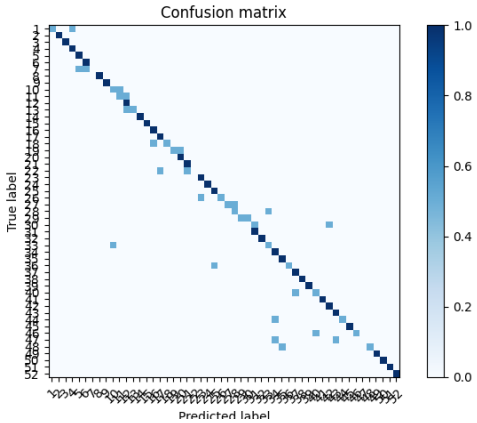



Figure 5. Confusion matrix for PCA-LDA

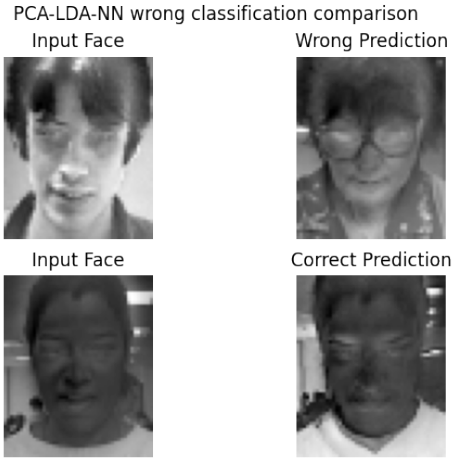



Figure 6. Correct and incorrect classification using PCA-LDA

### 4.1. Randomisation in feature space

This method takes a random-subset of the training data to train each classifier. The randomness parameter is the

number of samples for each model. However, if the number of samples is large, there will be a higher correlation between the classifiers. So it is important to keep the training examples large enough but not too large so that they cause unnecessary correlation between the classifiers.

## 5. Image Categorization of Caltech 101

Now, the performance of the Random Forest classifier will be evaluated on a real dataset, the `Caltech 101` dataset. It consists of 10 classes, where we randomly choose 15 images per class for training and testing, respectively. Instead of raw pixels, SIFT transformations will be applied and extracted descriptors will be utilised. Then, a codebook is constructed via either k-Means clustering or Random Forest embeddings. The resulting histogram representations, Bag-of-(Visual)-Words (**BoW**), will be subsequently used as classification features.

### 5.1. *k*-means codebook

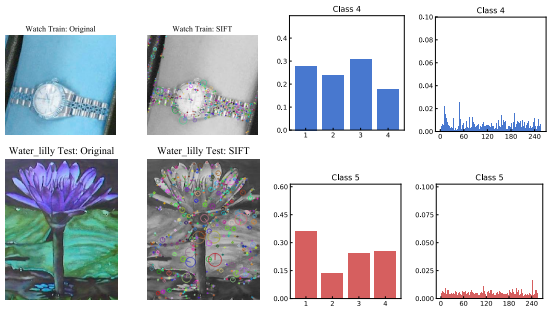
The codebook construction algorithm using *k*-Means clustering is comprehensively provided in Algorithm 2 of appendix section.

#### 5.1.1 Vector Quantization Process

In Figure 6, we present training and testing image examples from original, SIFT and Bag-of-Words representations. Interestingly, original images could possess differently varying sizes, distinctive heights and widths, whereas their correspondent BoW representations will be consistent across all pictures, enabling to utilize a single classifier for all image dataset. The precise dimensionality of the BoW representation will be managed by the number of centroids (**vocabulary size**) which are used for *k*-means clustering.

SIFT is particularly used for descriptor extraction. As observed from the Figure 7b, SIFT is applied to gray-scale images, which returns 128-dimensional descriptors. Using the training dataset pictures, we can collect all the descriptors and uniformly sample $100K$ of them without replacement, where those will be used for training a *k*-means clustering model.

During the BoW modelling, codewords (i.e. centroids) are given by the vocabulary, which eliminates the information loss introduced by clustering in the visual code-book construction. Thus, after measuring the cluster centers, **vector quantization** can be done to retrieve the BoW representation. We assign to every descriptor in an image the nearest centroid obtained from the *k*-means clustering. We then count the occurrences of each centroid, which ultimately produces the BoW histogram representation.




(a) Original (b) SIFT (c) $K = 4$ (d) $K = 256$

Figure 7. Training (top) and testing (bottom) images. (a) Original image and (b) SIFT descriptors (c) BoW using $K = 4$ (d) BoW using $K = 256$

#### 5.1.2 Vocabulary size

As a result, the Bag-of-Word representation is highly dependent on the **vocabulary size**, *k*. If *k* is chosen too small, generalization will be poor as there will not be enough discrimination between the various classes (underfitting). On the other hand, if *k* is too large, generalization may improve at the cost of computational performance. Thus, the choice of *k* is a hyperparameter that should be validated. In Figure 7c, 7d the BoW representation of a train and test example is provided for $k = 4$ and $k = 256$, showcasing the conspicuous difference between 2 feature vectors for different vocabulary sizes.

### 5.2. Random Forest Classifier

Establishing the transformation from pixels to Bag-of-Word indeed allows us to train the Random Forest and test its performance properly. Given the 10 classes, we train the Random Forest classifier using the BoW-represented training images. Certainly, as observed from lecture discussions, there are a variety of hyperparameters that strongly influence the classification accuracy of the model. In the following sub-discussions, we will mention the considerable effects of these hyperparameters on the model and their optimal values. The selection of the hyperparameters, though, will be decided based on the cross-validation error, accompanying training and testing errors for illustrating comparisons.

#### 5.2.1 Maximum Tree Depth

As discussed in the previous section and lectures, we will consider the effects of maximum tree depth on classification. Figure 8 indicates the accuracy and timing performance of the model for different values of **max_depth**. In regards of the accuracy, classification is compromised at both extreme cases; for smaller values, the Random Forest underfits the data because of an inability to split the dataset

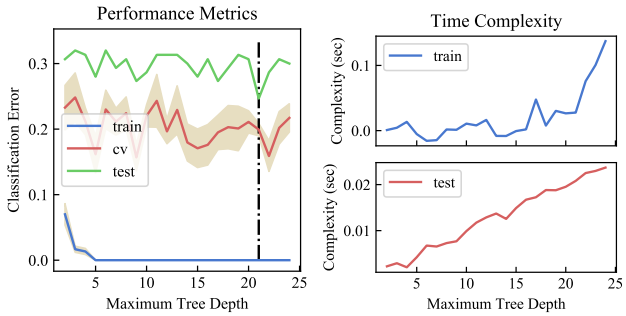



Figure 8. Classification accuracy (left) and time complexity (right) for various tree depths

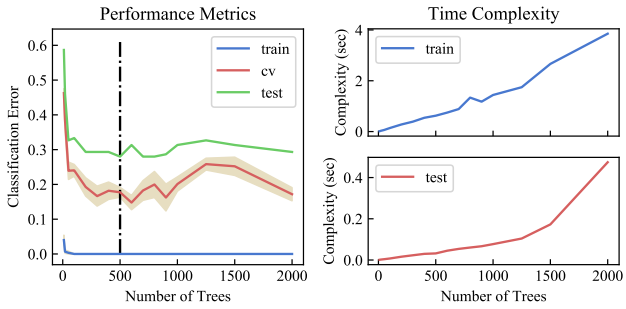



Figure 9. Classification accuracy (left) and time complexity (right) for various number of trees

effectively. However, regarding the larger values, Random Forest begins to overfit since the leaf nodes possess fewer points. Concerning the efficiency of time, it is unsurprising to mention how the timing increases for higher values of max_depth, both for training and testing steps. Thus, as observed from the Figure 8, we may choose **max_depth = 14** to obtain considerably good generalization and timing performance for our model.

#### 5.2.2 Number of Trees

In the Figure 9, we will consider how the number of trees impacts our model. As learned from lecture contents, the classification accuracy starts to increase with an increasing number of trees, as individual tendencies of trees to overfit are averaged out by the Random Forest. Such an overfitting tendency is specifically dominant for lower numbers of trees, where no averaging effect could be observed. Nevertheless, time complexity will increase almost linearly with respect to the number of trees. Overall, taking into account steepness and following plateau of illustrated curves, we may conclude to choose **num_trees = 900**, accomplishing decent timing performance both for training/testing with a minor accuracy decrease.

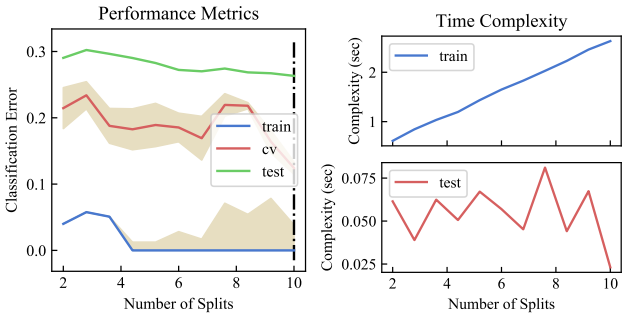



Figure 10. Classification accuracy (left) and time complexity (right) for various number of splits (degree of randomness)

#### 5.2.3 Degree of Randomness and Weak Learners

It is well-known that degree of randomness is managed by the number of random splits. In Figure 10, we can observe how the accuracy improves whenever the number of splits starts to increase, whilst plateauing at **num_splits = 7**, in which training complexity increases linearly.

Axis-aligned, two-pixel test, linear, quadratic and cubic kernels are taken into account for node splitting procedure. Figure 11 showcases the considerable effect weak learner has on accuracy and timing complexity. Distinctively from the figure, one may conclude that best cross-validation accuracy is achieved for the two-pixel learner, in which training and testing accuracy remain considerably stable. More interestingly, though, the computational timing performance of the model starts to deteriorate even further with more complicated models - which was expected as such models are fundamentally more computationally intensive if compared to their simpler ones. Therefore, we conclude to select two-pixel weak learners for our model.

#### 5.2.4 Vocabulary Size

One of the particular relevant aspects for the image classification problem is the vocabulary size that Bag-of-Words implementation used. Figure 12 provides accuracy and timing features of the vocabulary size. We may notice how an initial steep decrease occurs for increasing vocabulary size smaller than approximately 48 50. Also mentioned from previous discussions, underfitting occurs because of discrimination lacking between different classes. However, after the approximate size of 50, a gradual increase for higher sizes occurs in both cross-validation and test errors. Such phenomena could occur due to the tendency of Random Forest's overfitting in the larger dimensionality.

#### 5.2.5 Optimal Configuration

The selection of the optimal hyperparameters is summarized in Table 3. The selected model achieves 83.4% accuracy on the test set, and its confusion matrix and suc-

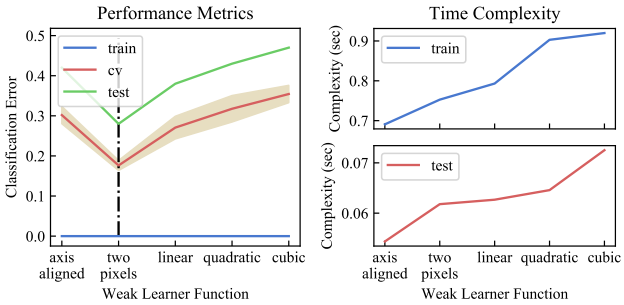



Figure 11. Accuracy and timing for various weak learners

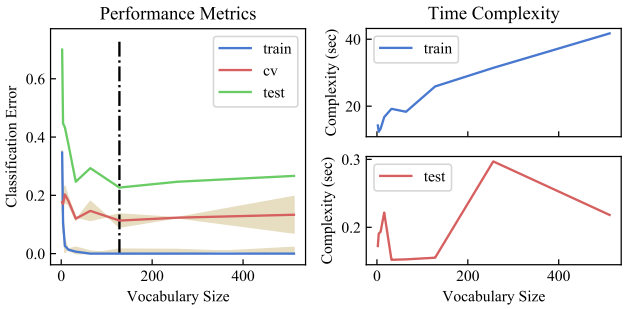



Figure 12. Classification accuracy (left) and time complexity (right) for various vocabulary sizes

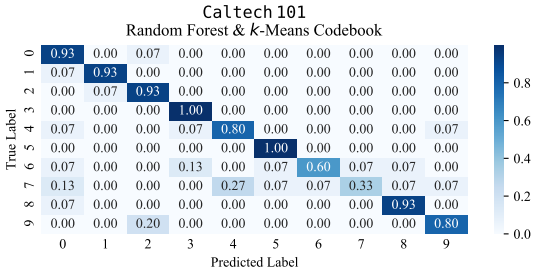



Figure 13. Confusion Matrix: RF Classifier on $k$-Means codebook

cess/failure examples of classification are provided in Figures 13 and 14, respectively.

According to the confusion matrix, classifier poorly recognizes watches, treating them as wheelchairs or ticks, intuitively because of their round shapes.

## 5.3. Random Forest Codebook

### 5.3.1 Vector Quantisation Process

During the final section, we will build a Random Forest Codebook from the SIFT descriptors of the images. Different from $k$-means, the RF codebook exploits the valuable discriminative information in the descriptor space, which is constructed in a supervised fashion.

During the training process, labels are considered for the Random Forest construction and descriptors are correspondingly assigned to "cluster", which is determined by the leaves they end up in. Every descriptor reaches $T$ leaves for a Random Forest with $T$ Decision Trees. This is an encoding procedure, in which the descriptor space is mapped to leaf embeddings. The Bag of Words representation is then determined by collecting all leaf embeddings of the image, after transforming all of its SIFT descriptors. The precise number of leaves, or **vocabulary size** $K$, of the Random Forest codebook is not directly controlled, and merely an upper bound is found: $K \leq num\_trees \times 2^{max\_depth-1}$


Figure 14. Random Forest Classifier on $k$-Means codebook: (top) examples of worst accuracy on "watch" class, (bottom) examples of correct classifications of respective classes

### 5.3.2 Optimal Configuration

There are currently 2 Random Forest models; one feature transformer (codebook) and one classifier, in which both of these models' hyperparameters need to be tuned, similar to those discussions in section 3.2. Cross-Validation will be used again to optimize the desired hyperparameters. Testing all feasible combinations of hyperparameters for those 2 models, the Table 3 illustrates optimal model parameters.

The model obtains 77.3% classification accuracy on the test set, where umbrella images are poorly classified as shown in Figures 16 and 18. As a sideline, it is noteworthy to mention how prediction bias occurs towards ying-yang images (class 5), where majority of wrongly classified pictures are labeled as class 5 in our model.

### 5.3.3 Comparison

Now, we compare the 2 Bag-of-words implementations. The desired comparison matrix is finalized in Table 3. From the accuracy perspective, $k$-Means codebook results in a better recognition accuracy, 83.4% over 77.3% for RF. However, compressed and rich discriminative information which Random Forest codebook encodes culminates in a remarkably faster classifier, in comparison to the one developed with $k$-Means codebook. Specifically, the Random Forest codebook uses an optimal 200 DTs, with maximum depth 7 RF, whereas the $k$-means uses an optimal 900 DTs and maximum depth 14 RF. As a sideline, we note how

the RF codebook scales much efficiently than the $k$-Means counterpart, as indicated in Figure 17.

## References


[1] Mathematical Foundations, R.Duda, P.Hart, D.Stork, Pattern Classification (Second Edition), JOHN WILEY SONS, Inc. 2001

[2] D. Arthur and S. Vassilvitskii. k-means ++ : The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027 – 1035. Society for Industrial and Applied Mathematics, 2007.

[3] D. Sculley. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*, ACM, 2010.


## Appendix

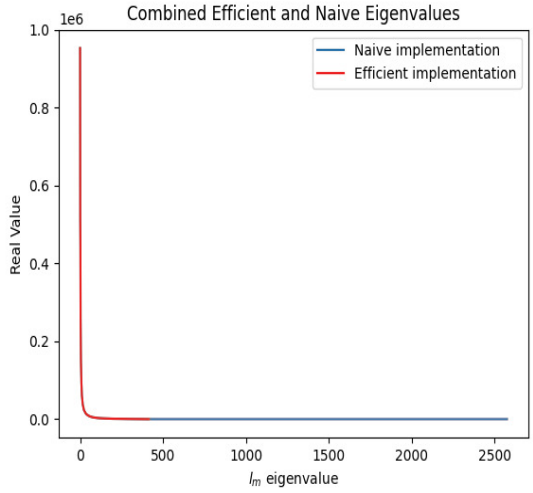



Figure 15. Same eigenvalues from naive PCA and efficient PCA

| Caltech 101 Dataset: Random Forest Models | | | |
|---|---|---|---|
| **Codebook** | Algorithm | $k$-Means | RF |
| | Number of Centroids ($k$) | 128 | - |
| | Number of Trees | - | 10 |
| | Weak Learner | - | two-pixel |
| | Maximum Tree Depth | - | 5 |
| | Number of Splits | - | 5 |
| | Vocabulary Size | 128 | ≈204 |
| **Classifier** | Weak Learner | two-pixel | two-pixel |
| | Maximum Tree Depth | 14 | 7 |
| | Number of Trees | 900 | 200 |
| | Minimum Samples at Node | 5 | 5 |
| | Number of Splits | 7 | 7 |
| **Score** | Training Accuracy | 100 % | 100 % |
| | Validation Accuracy | 86 % | 80.6 % |
| | Testing Accuracy | 83.4 % | 77.3 % |

Table 3. Random Forest model hyperparameters and performance summary on **Caltech 101** dataset

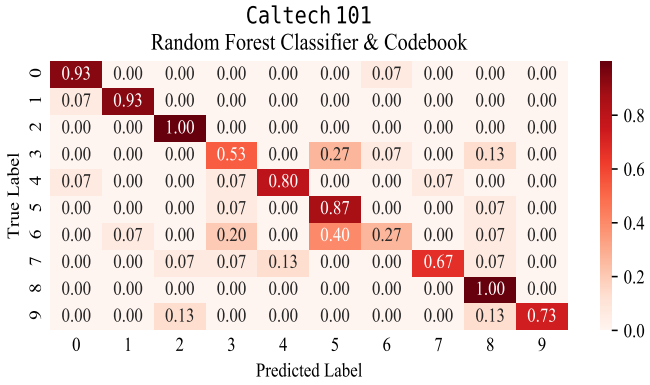



Figure 16. Confusion Matrix: RF Classifier on Random Forest codebook

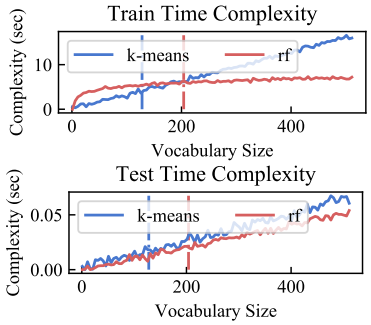



Figure 17. Vector Quantisation Process Complexity comparison for $k$-Means and RF Codebook


Figure 18. Random Forest Classifier on Random Forest codebook: (top) examples of worst accuracy on "umbrella" class, (bottom) examples of correct classifications of respective classes

**Algorithm 2:** $k$-Means codebook generation

**input :** vocab size $K$
number of descriptors $N$
training & testing images, $S_{\text{train}}$ & $S_{\text{test}}$
**output:** BoW representation of $S_{\text{train}}$ & $S_{\text{test}}$

initialize $\mathcal{D}$ as empty set;
**foreach** $\mathcal{I}_i \in S_{\text{train}}$ **do**
  extract SIFT descriptors for $\mathcal{I}_i$ and add to $\mathcal{D}$
**end**
pick $N$ randomly selected descriptors from $\mathcal{D}$;
train $k$-means on chosen descriptors, with $k = K$;
**foreach** $\mathcal{I}_i \in S_{\text{train}} \cup S_{\text{test}}$ **do**
  transform $\mathcal{I}_i$ from pixels to BoW representation using trained $k$-means
**end**

Figure 19. Codebook Construction using $k$-Means clustering

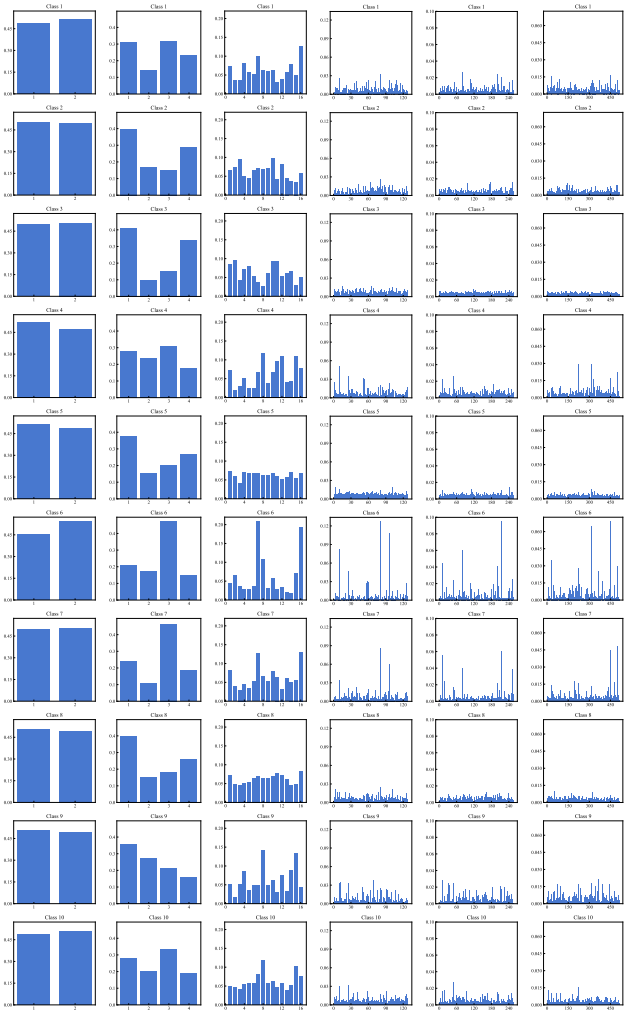


Figure 20. $k$-Means codebook centroids BoW representation (training):
(a) $K = 2$ (b) $K = 4$ (c) $K = 16$
(d) $K = 128$ (e) $K = 256$ (c) $K = 512$

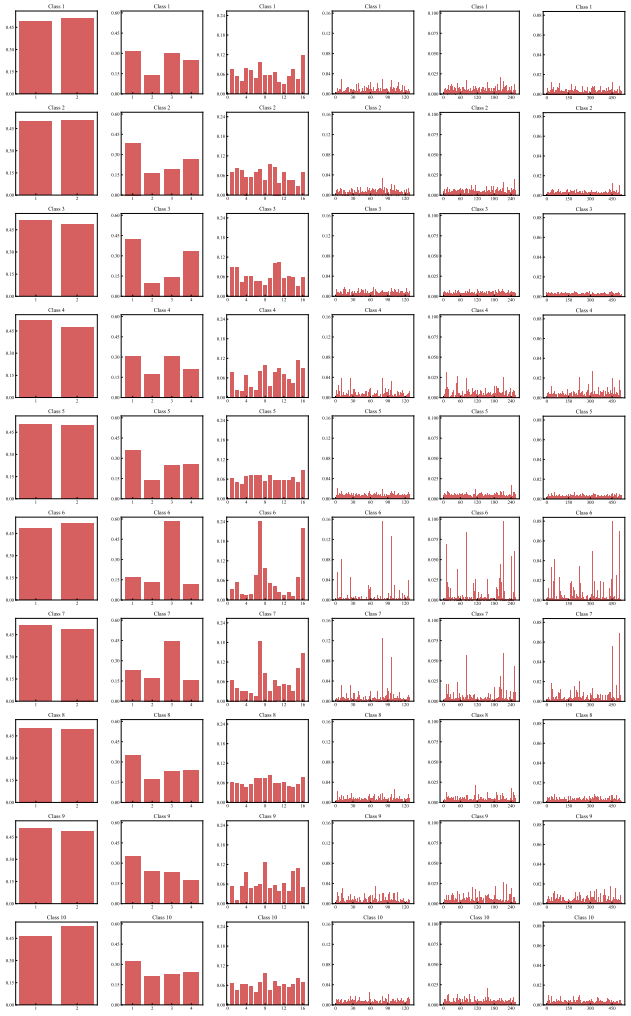


Figure 21. *k*-Means codebook centroids BoW representation (testing):

(a) $K = 2$ (b) $K = 4$ (c) $K = 16$
(d) $K = 128$ (e) $K = 256$ (c) $K = 512$

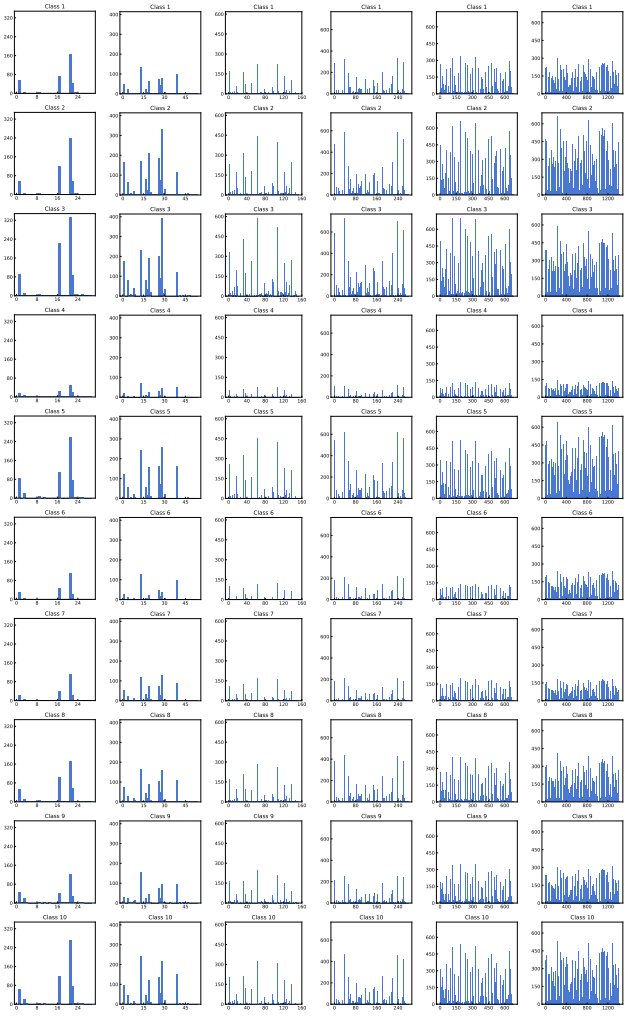


Figure 22. RF codebook BoW representation (training):
(a) **num_trees** $= 1$ (b) **num_trees** $= 2$ (c) **num_trees** $= 5$
(d) **num_trees** $= 10$ (e) **num_trees** $= 25$ (f) **num_trees** $= 50$

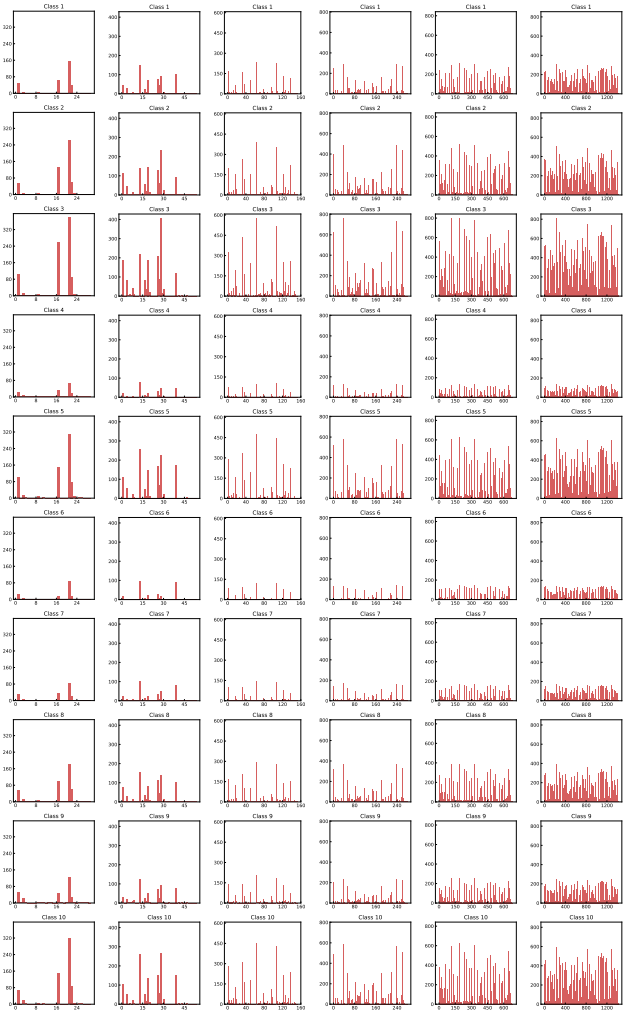


Figure 23. RF codebook BoW representation (testing): (a) **num_trees** $= 1$ (b) **num_trees** $= 2$ (c) **num_trees** $= 5$ (d) **num_trees** $= 10$ (e) **num_trees** $= 25$ (f) **num_trees** $= 50$