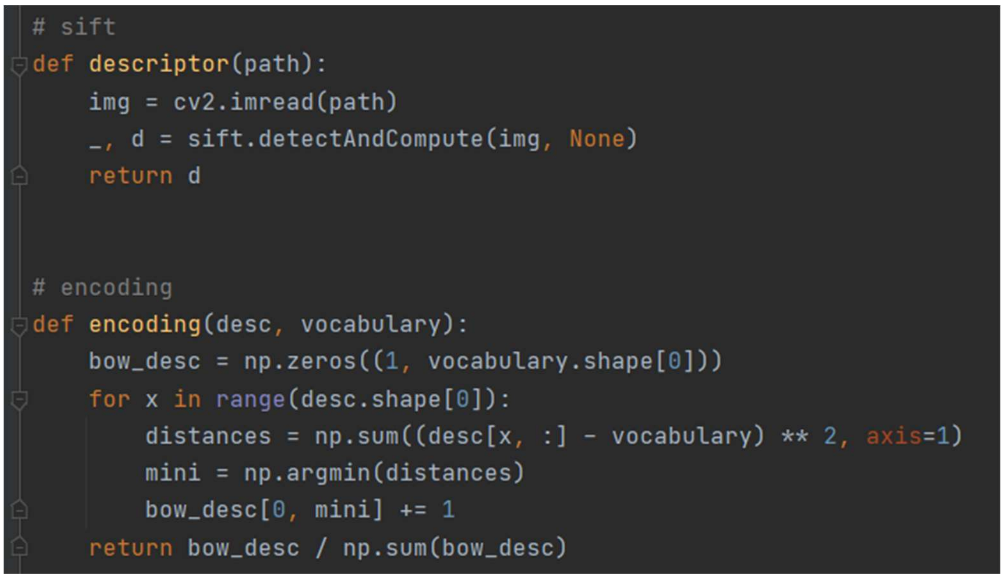
Computer Vision: Technical Report of 3rd Project

Papadopoulos Aristeidis A.M.:57576



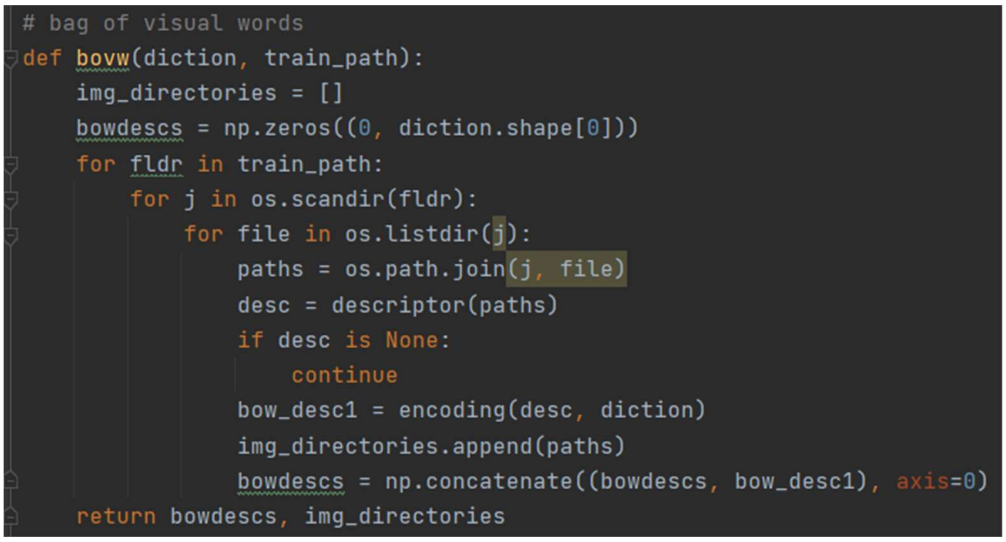
```python
# sift
def descriptor(path):
    img = cv2.imread(path)
    _, d = sift.detectAndCompute(img, None)
    return d


# encoding
def encoding(desc, vocabulary):
    bow_desc = np.zeros((1, vocabulary.shape[0]))
    for x in range(desc.shape[0]):
        distances = np.sum((desc[x, :] - vocabulary) ** 2, axis=1)
        mini = np.argmin(distances)
        bow_desc[0, mini] += 1
    return bow_desc / np.sum(bow_desc)
```

Image 1. Function declarations used for feature extraction and image encoding.

- ## Bag Of Visual Words (BOVW) - Encoding:

The dictionary used is based on the Bag Of Visual Words model, using K-means clustering. In the BOVW model, each image is represented by a selection of features, created by K-means clustering, where K is the number of groups and with each group representing one distinct feature.



```python
# bag of visual words
def bovw(diction, train_path):
    img_directories = []
    bowdescs = np.zeros((0, diction.shape[0]))
    for fldr in train_path:
        for j in os.scandir(fldr):
            for file in os.listdir(j):
                paths = os.path.join(j, file)
                desc = descriptor(paths)
                if desc is None:
                    continue
                bow_desc1 = encoding(desc, diction)
                img_directories.append(paths)
                bowdescs = np.concatenate((bowdescs, bow_desc1), axis=0)
    return bowdescs, img_directories
```

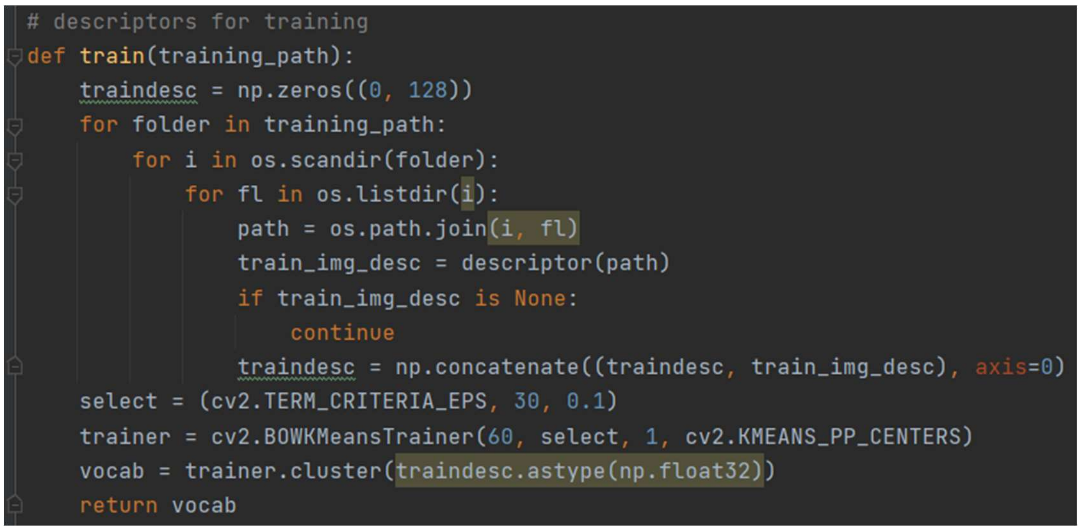
Image 2. Implementation of Bag of Visual Words model.

The features are detected by using descriptors and keypoints, found by the SIFT algorithm. Then, K-means clustering is executed, consisting of the following steps:

1. Each vector created by SIFT descriptor, is placed into the hyperplane, with dimensions corresponding to its. More specifically, the dimension

here is 128. This means that each point in space is described by 128 coordinates.

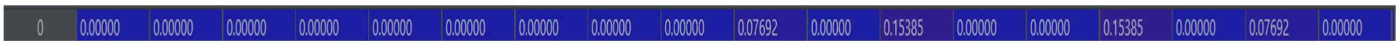
2. Vectors placed in similar positions are grouped, with each group referring to a feature for those images.
3. Grouping of vectors is described in the following steps. Firstly, random points, K in number, are used as centers for each group-class.
4. Distances of each vector from those centers are calculated and grouping is made based on those k minimum distances.
5. The medians of those distances are calculated for each class and based on those, new group centers are selected for each group and the process is repeated.
6. When the medians are stabilized, the process terminates.



```python
# descriptors for training
def train(training_path):
    traindesc = np.zeros((0, 128))
    for folder in training_path:
        for i in os.scandir(folder):
            for fl in os.listdir(i):
                path = os.path.join(i, fl)
                train_img_desc = descriptor(path)
                if train_img_desc is None:
                    continue
                traindesc = np.concatenate((traindesc, train_img_desc), axis=0)
    select = (cv2.TERM_CRITERIA_EPS, 30, 0.1)
    trainer = cv2.BOWKMeansTrainer(60, select, 1, cv2.KMEANS_PP_CENTERS)
    vocab = trainer.cluster(traindesc.astype(np.float32))
    return vocab
```

Image 3. Declaration of training function.

Now, it is possible to describe each picture with a vector of specific length and the encoding of each image in a group-class, based on this vector. Furthermore, each image's features are found and then the times they appear in the dictionary is counted. The result is a matrix where its rows are the images in the dataset and its columns the features used to describe each image, i.e. the number of clusters, equal to K. The content of the encoding can be understood by the following example:

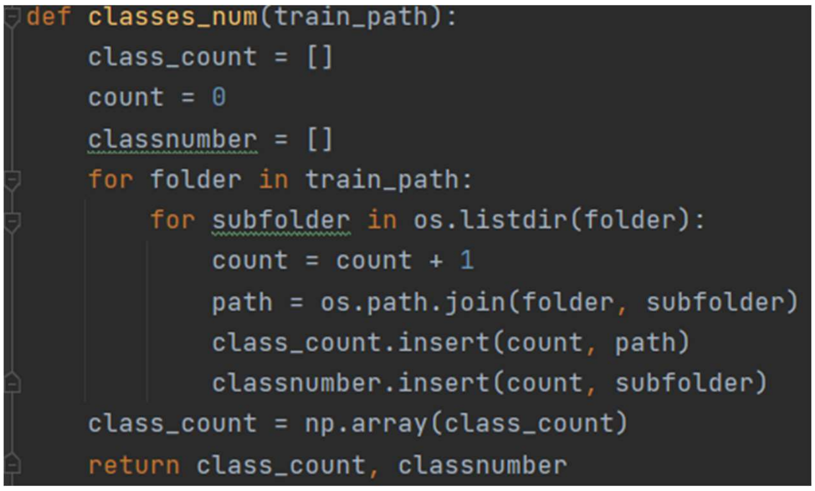


0	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.07692	0.00000	0.15385	0.00000	0.00000	0.15385	0.00000	0.07692	0.00000

Image 4. Result of the encoding for the first image of the train dataset.

In each cell is depicted how many times (how intense) each feature is in the first image of the dataset. K was selected to be 60, so the dictionary is 60 features long. The number was selected arbitrary.

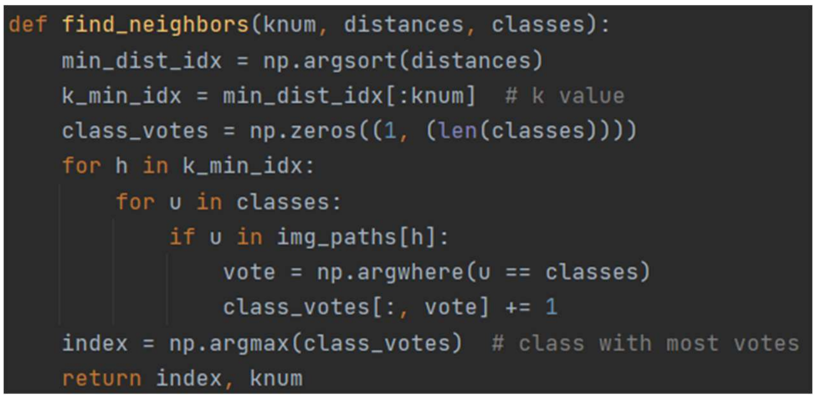
- **Knn (K nearest neighbors)**



```python
def classes_num(train_path):
    class_count = []
    count = 0
    classnumber = []
    for folder in train_path:
        for subfolder in os.listdir(folder):
            count = count + 1
            path = os.path.join(folder, subfolder)
            class_count.insert(count, path)
            classnumber.insert(count, subfolder)
    class_count = np.array(class_count)
    return class_count, classnumber
```

Image 5. Declaration of the function used to find the number of classes and their directory.

Firstly, the number of classes must be found, in order to know where to classify each image. This can be achieved by the function that can be seen in the above image, where each class name is placed in a list, along with its corresponding path. In order to use Knn algorithm, images in test set must go through previous steps, so that their features are found and their encoding made according to BOVW model. Now, for each image in the test dataset, the k nearest neighbors must be found, where the term nearest refers to the neighbors with the minimum Euclidean Distance. Then, the class where most of the k neighbors belong is assigned to each image. The algorithm can be seen in the following image:

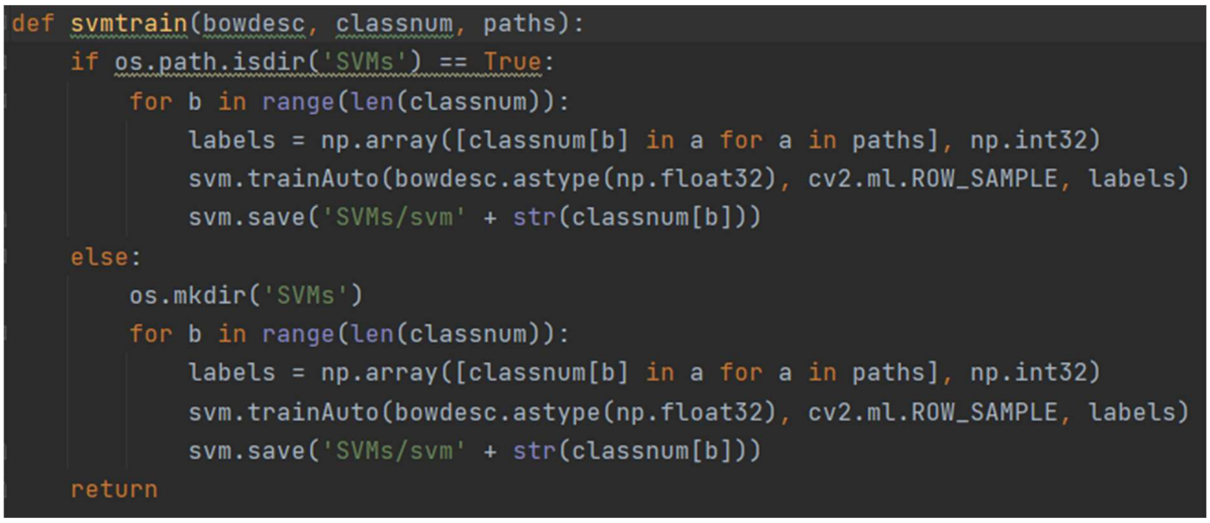


```python
def find_neighbors(knum, distances, classes):
    min_dist_idx = np.argsort(distances)
    k_min_idx = min_dist_idx[:knum]  # k value
    class_votes = np.zeros((1, (len(classes))))
    for h in k_min_idx:
        for u in classes:
            if u in img_paths[h]:
                vote = np.argwhere(u == classes)
                class_votes[:, vote] += 1
    index = np.argmax(class_votes)  # class with most votes
    return index, knum
```

Image 6. Knn implementation

After the distances are calculated, they are sorted by the index of the minimum distances and then the K minimum ones are selected. Then, for each index selected, a search is made, in order to find the class where each image belongs in, and a vote is "casted" in the corresponding index of the list that holds the votes. The index with the most votes, that's where the image is classified. By finding that index, the class where each image is classified can be found.

- **Support Vector Machine (SVM) – Multi-class Classification**

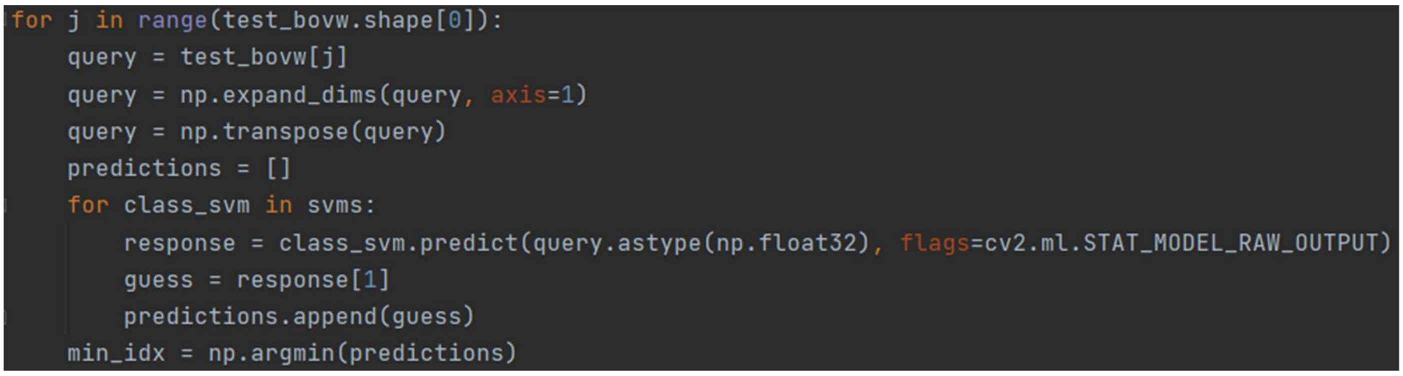


```python
def svmtrain(bowdesc, classnum, paths):
    if os.path.isdir('SVMs') == True:
        for b in range(len(classnum)):
            labels = np.array([classnum[b] in a for a in paths], np.int32)
            svm.trainAuto(bowdesc.astype(np.float32), cv2.ml.ROW_SAMPLE, labels)
            svm.save('SVMs/svm' + str(classnum[b]))
    else:
        os.mkdir('SVMs')
        for b in range(len(classnum)):
            labels = np.array([classnum[b] in a for a in paths], np.int32)
            svm.trainAuto(bowdesc.astype(np.float32), cv2.ml.ROW_SAMPLE, labels)
            svm.save('SVMs/svm' + str(classnum[b]))
    return
```

Image 7. Declaration of the function that utilized SVM multiclass classification.

The above function implements the SVM classification algorithm, where firstly a check is made, whether or not the directory to save SVMs exists. Then, the classification is made based on these steps:

1. The hyperplane where images are projected (by using descriptors) is split into two half-planes, where on the one side images of one class exist, and in the other half all other images exist. The line that splits the two half-planes is selected in a way that the distance between the line and the samples is maximized.
2. Code-wise, step one is implemented as follows: The matrix named labels has indexes as many as the images in the training dataset, where "1" is placed in the index where the image in question is examined and "0" in every other index, for all the other classes. So, the matrix labels is created as many times as the classes that have been found and similarly that many SVMs are created.
3. Using every SVM created, for each image of the test dataset, in order to find the SVM that minimizes the distance between each test image and the line that splits the half-planes, for every SVM. The index with the minimum distance corresponds to the class that the test image is classified.

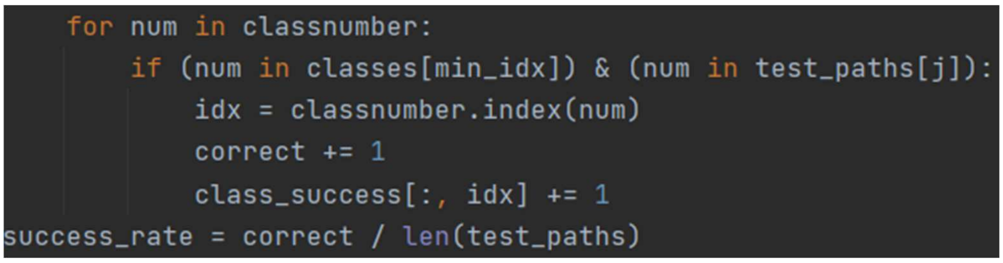


```
for j in range(test_bovw.shape[0]):
    query = test_bovw[j]
    query = np.expand_dims(query, axis=1)
    query = np.transpose(query)
    predictions = []
    for class_svm in svms:
        response = class_svm.predict(query.astype(np.float32), flags=cv2.ml.STAT_MODEL_RAW_OUTPUT)
        guess = response[1]
        predictions.append(guess)
    min_idx = np.argmin(predictions)
```

Image 8. Implementation of SVM classification.

- **Evaluation method for each classification.**

The evaluation method for each classification can be seen in the following image:



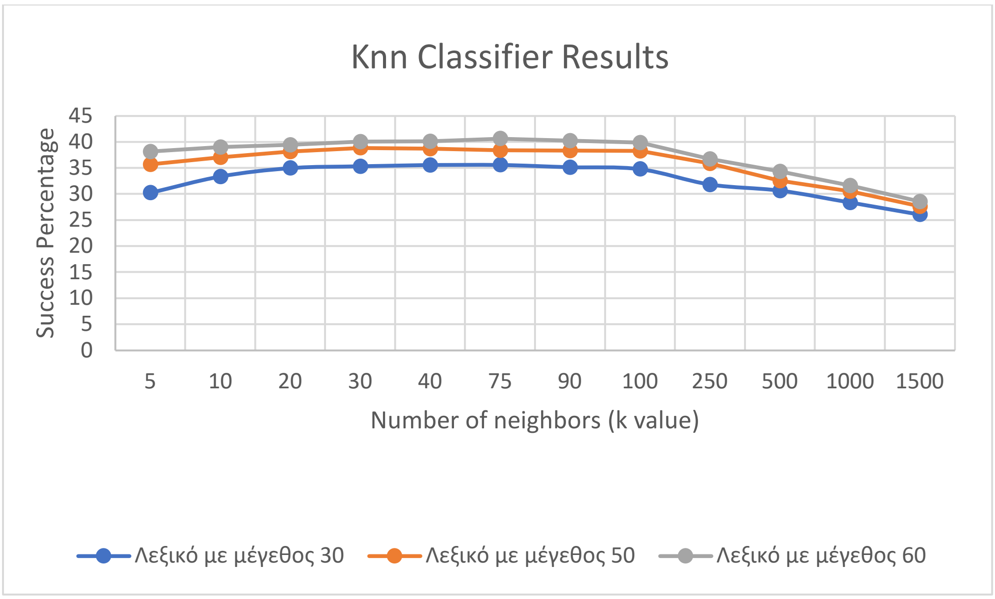
```
    for num in classnumber:
        if (num in classes[min_idx]) & (num in test_paths[j]):
            idx = classnumber.index(num)
            correct += 1
            class_success[:, idx] += 1
success_rate = correct / len(test_paths)
```

Image 9. Evaluation function for each method of classification

For each class predicted, a check is made if the correct class was found. If that is the case, a counter is increased and a corresponding index is also increased. Below, the results can be seen for using Knn algorithm, for different dictionary lengths and for different K number.

- Conclusions for using Knn classifier:

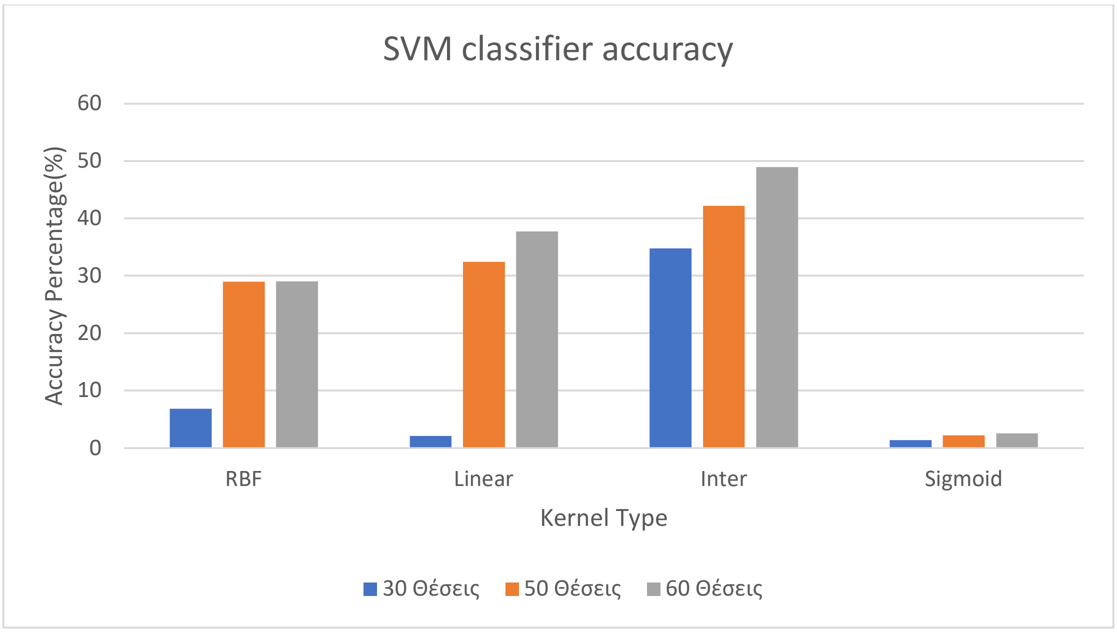
K (number of neighbors)	Accuracy Percentage for dictionary of 30 features	Accuracy Percentage for dictionary of 50 features	Accuracy Percentage for dictionary of 60 features
5	30,24	35,69	38,11
10	33,36	37,04	38,99
20	34,99	38,11	39,41
30	35,32	38,85	40,02
40	35,55	38,71	40,07
75	35,6	38,39	40,62
90	35,13	38,29	40,2
100	34,80	38,25	39,83
250	31,83	35,87	36,71
500	30,66	32,52	34,3
1000	28,38	30,52	31,6
1500	26,06	27,64	28,57



Considering the above results, it can easily be understood that the number of correct classifications is highly dependent on the number of K, as the success rate is increased up to a point and then it is decreased to very low percentages. Furthermore, for k values ranging from 30 to 100, the success rate seems to stabilize. For k values over 1500, the method was not used, as its effectiveness is lost and the computational time is increased highly. This is due to the fact that a low value of k is not enough for a successful classification, while for a high value of k overcomplicates this method and success is only available for classes with a high number of image samples. What's more, the importance of the length of the BOVW dictionary can be understood, since for the same value of k, the higher the dictionary is, the higher the accuracy percentage is. So, by representing the images with a higher number of features, the images are represented more accurately. Of course, the dictionary could contain all the features that are extracted from SIFT descriptor(128 in number), but in this case a lot of computational resources are used, while the success of the aforementioned methods is in question.

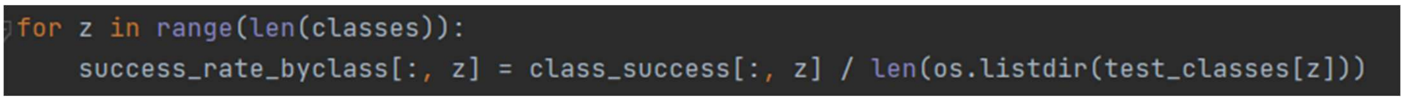
- Conclusions for using SVM classifier:

Kernel Type	**Accuracy Percentage for dictionary of 30 features**	**Accuracy Percentage for dictionary of 50 features**	**Accuracy Percentage for dictionary of 60 features**
RBF	6,84	28,94	28,99
Linear	2,09	32,39	37,69
Inter	34,76	42,16	48,91
Sigmoid	1,35	2,19	2,56



In accordance with the above, and with what was mentioned about the dictionary length, the results of SVM classification are identical to those of knn classification. The higher the dictionary is, the better the results are, for a specific kind of kernel. Furthermore, it can be observed that the best type of kernel is the Histogram Intersection Kernel, with accuracy nearing 50%, while the worst kernel type is Sigmoid, whose accuracy is under 3%. Histogram Intersection Kernel seems to be more accurate due to the fact that this kernel is based on the presence of colors and their corresponding histograms. Since the images in this project are traffic signs, with strong color presence, so it makes sense that this kind of kernel yields successful results. For the same reason, there is a high difference in the results between this kind of kernel and the rest. Finally, for a dictionary with length of 30, all kernels yield low accuracy percentage, except from Histogram Intersection Kernel.

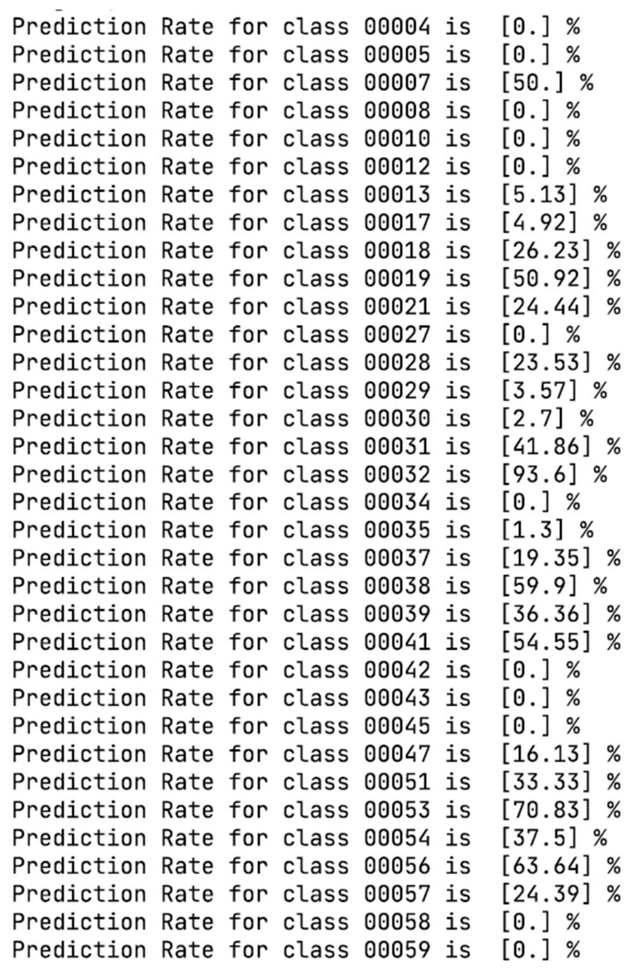
- **Classification results evaluation of each method per class**



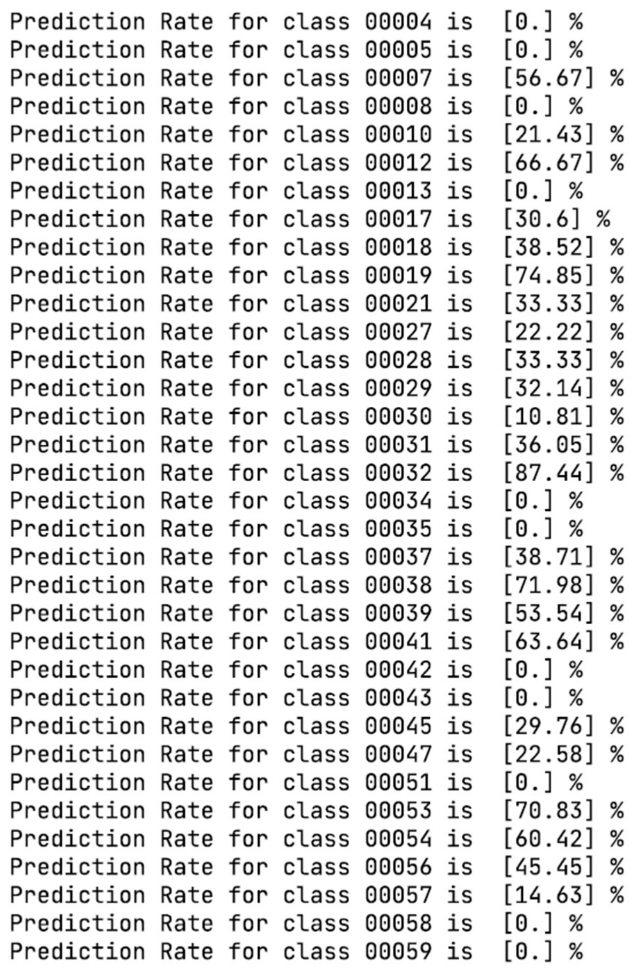
```
for z in range(len(classes)):
    success_rate_byclass[:, z] = class_success[:, z] / len(os.listdir(test_classes[z]))
```

Image 10. Evaluation of classification results per class.

The accuracy percentage per class is calculated by dividing the successful classifications for a given class with the number of images from the test set. Indicatively, the results per class are shown in the following images, for a dictionary with length of 60 indexes, first with k value equal to 75 and then with SVM, using as kernel the Histogram Intersection Kernel.



```
Prediction Rate for class 00004 is  [0.] %
Prediction Rate for class 00005 is  [0.] %
Prediction Rate for class 00007 is  [50.] %
Prediction Rate for class 00008 is  [0.] %
Prediction Rate for class 00010 is  [0.] %
Prediction Rate for class 00012 is  [0.] %
Prediction Rate for class 00013 is  [5.13] %
Prediction Rate for class 00017 is  [4.92] %
Prediction Rate for class 00018 is  [26.23] %
Prediction Rate for class 00019 is  [50.92] %
Prediction Rate for class 00021 is  [24.44] %
Prediction Rate for class 00027 is  [0.] %
Prediction Rate for class 00028 is  [23.53] %
Prediction Rate for class 00029 is  [3.57] %
Prediction Rate for class 00030 is  [2.7] %
Prediction Rate for class 00031 is  [41.86] %
Prediction Rate for class 00032 is  [93.6] %
Prediction Rate for class 00034 is  [0.] %
Prediction Rate for class 00035 is  [1.3] %
Prediction Rate for class 00037 is  [19.35] %
Prediction Rate for class 00038 is  [59.9] %
Prediction Rate for class 00039 is  [36.36] %
Prediction Rate for class 00041 is  [54.55] %
Prediction Rate for class 00042 is  [0.] %
Prediction Rate for class 00043 is  [0.] %
Prediction Rate for class 00045 is  [0.] %
Prediction Rate for class 00047 is  [16.13] %
Prediction Rate for class 00051 is  [33.33] %
Prediction Rate for class 00053 is  [70.83] %
Prediction Rate for class 00054 is  [37.5] %
Prediction Rate for class 00056 is  [63.64] %
Prediction Rate for class 00057 is  [24.39] %
Prediction Rate for class 00058 is  [0.] %
Prediction Rate for class 00059 is  [0.] %
```



```
Prediction Rate for class 00004 is  [0.] %
Prediction Rate for class 00005 is  [0.] %
Prediction Rate for class 00007 is  [56.67] %
Prediction Rate for class 00008 is  [0.] %
Prediction Rate for class 00010 is  [21.43] %
Prediction Rate for class 00012 is  [66.67] %
Prediction Rate for class 00013 is  [0.] %
Prediction Rate for class 00017 is  [30.6] %
Prediction Rate for class 00018 is  [38.52] %
Prediction Rate for class 00019 is  [74.85] %
Prediction Rate for class 00021 is  [33.33] %
Prediction Rate for class 00027 is  [22.22] %
Prediction Rate for class 00028 is  [33.33] %
Prediction Rate for class 00029 is  [32.14] %
Prediction Rate for class 00030 is  [10.81] %
Prediction Rate for class 00031 is  [36.05] %
Prediction Rate for class 00032 is  [87.44] %
Prediction Rate for class 00034 is  [0.] %
Prediction Rate for class 00035 is  [0.] %
Prediction Rate for class 00037 is  [38.71] %
Prediction Rate for class 00038 is  [71.98] %
Prediction Rate for class 00039 is  [53.54] %
Prediction Rate for class 00041 is  [63.64] %
Prediction Rate for class 00042 is  [0.] %
Prediction Rate for class 00043 is  [0.] %
Prediction Rate for class 00045 is  [29.76] %
Prediction Rate for class 00047 is  [22.58] %
Prediction Rate for class 00051 is  [0.] %
Prediction Rate for class 00053 is  [70.83] %
Prediction Rate for class 00054 is  [60.42] %
Prediction Rate for class 00056 is  [45.45] %
Prediction Rate for class 00057 is  [14.63] %
Prediction Rate for class 00058 is  [0.] %
Prediction Rate for class 00059 is  [0.] %
```

Images 11,12. Percentages of successful classifications for each class, using Knn (left) and SVM (right).

SVM classifier seems more variant in its accuracies, as it is more accurate in different classes than knn. Furthermore, classes with multiple training images are more accurately classified than those with less. For example, class "00032", has 93.6% accuracy using Knn classifier, while with SVM classifier the accuracy is 87.44%.

Image 13. The first image of training dataset for class "00032".

On the contrary, classes with low number of images in the training dataset have a low count or even zero correct classifications in both classifiers. The following classes are mentioned as examples:

Κλάση	**Ποσοστό με Knn(%)**	**Ποσοστό με SVM(%)**
00005	0	0
00051	33,33	0

Images 14,15. The first images of training dataset for classes “00005” and “00051” accordingly.

Furthermore, classes representing traffic signs with one color present, such as the following example, are less accurately classified.Επιπλέον, κλάσεις με σήματα που είναι μονοχρωματικά, όπως το παράδειγμα που ακολουθεί, παρουσιάζουν εξίσου χαμηλή απόδοση. More specifically, class “00034” is wrongly classified with both classifiers.

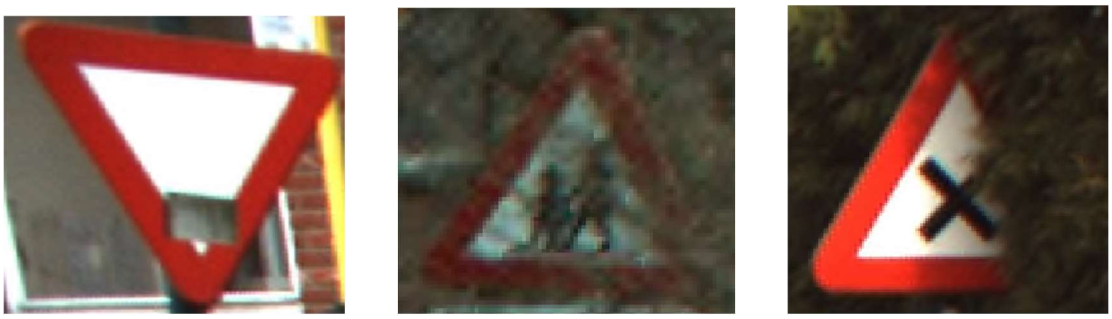
Image 16. The first image of training dataset for class “00034”.

Additionally, traffic signs depicting similar objects, also have low accuracy, while between them, the highest accuracy is achieved at the class with the highest count of training samples. For example, classes “00007”,”00008” and “00010” depict traffic signs with the presence of a human, with the most accurate classifications being at class “00007”, the class with the most training samples out of the three.

Class	**Accuracy using Knn(%)**	**Accuracy using SVM(%)**	**Number of images in train dataset**
00007	50	56,67	157
00008	0	0	27
00010	0	21,43	21

Images 17,18,19. The first images from training dataset for classes “00007”,”00008” and “00010”.

Finally, some images in the test dataset are contaminated by their surrounding environment, for example by tree leaves or stickers on top of them. So their classification is difficult, since they are affected by external factors and resulting in high error probability.



Images 20,21,22. Examples of images affected by external factors.

- **Final Results – Synopsis**

The current implementation, due to high usage of for-loops and if-statements, needs a lot of time in order to yield results. Partly, this is also due to the high number of images in both datasets. As always, a different implementation could result in different results. The SVM classifier, using Histogram Intersection Kernel returns moderately acceptable results, nearing 50% accuracy.

- **References**

- Richard Szeliski: Computer Vision Algorithms and Applications
- NumPy Documentation: https://numpy.org/doc/stable/contents.html
- OpenCV Documentation: https://docs.opencv.org/3.4/index.html
- John K. Tsotsos: Active Perception and Robot Vision: Indexing Via Color Histograms by M.J. Swain & D.H. Ballard