ΤΕΧΝΙΚΗ ΑΝΑΦΟΡΑ

`Για την Τρίτη Εργασία στο μάθημα

«Όραση Υπολογιστών»

Καθηγητής: Ιωάννης Πρατικάκης

Στέλιος Μούσλεχ

AM:57382

18/12/2019,Ξάνθη

ΕΙΣΑΓΩΓΗ

Στην παρακάτω τεχνική αναφορά θα περγραφεί η λύση της τρίτης εργασίας στο μάθημα «Όραση Υπολογιστών» αναλύοντας σε περιληπτικό επίπεδο το απαραίτητο θεωρητικό υπόβαθρο της επίλυσης καθώς θα γίνει και η του κώδικα και των αποτελεσμάτων

Τα ζητούμενα της Εργασίας ήταν να υλοποιηθεί πρόγραμμα σε Python με τη χρήση της βιβλιοθήκης OpenCV το οποίο θα αφορά στο πρόβλημα της ταξινόμησης πολλαπλών κλάσεων (multi-class classification). Το πρόγραμμα υλοποιείται με την εκτέλεση των παρακάτω βημάτων :

1. Παραγωγή οπτικού λεξικού (visual vocabulary) βασισμένη στο μοντέλο Bag of Visual Words(BOVW). Η δημιουργία του λεξικού να γίνει με τη χρήση του αλγορίθμου K-Meansχρησιμοποιώντας όλες τις εικόνες του συνόλου εκπαίδευσης (imagedb_train).

2. Εξαγωγή περιγραφέα σε κάθε εικόνα εκπαίδευσης (imagedb_train) με βάση το μοντέλο BOVW χρησιμοποιώντας το λεξικό που προέκυψε κατά το βήμα 1. Για το βήμα αυτό δεν μπορείτε ναχρησιμοποιήσετε τη σχετική κλάση της OpenCV (cv.BOWImgDescriptorExtractor).

3. Με βάση το αποτέλεσμα του βήματος 2, να υλοποιηθεί η λειτουργία ταξινόμησης μιας εικόναςκάνοντας χρήση των δυο παρακάτω ταξινομητών :

α. Του αλγορίθμου k-NN χωρίς τη χρήση της σχετικής OpenCV συνάρτησης(cv.ml.KNearest_create()).

β. Του σχήματος one-versus-all όπου για κάθε κλάση εκπαιδεύεται ένας SVM ταξινομητής.

4. Αξιολόγηση του συστήματος: Χρησιμοποιώντας το σύνολο δοκιμής (imagedb_test), να μετρηθεί η ακρίβεια του συστήματος (και στις δύο περιπτώσεις ταξινομητών) που εκφράζεται ως το ποσοστό των επιτυχών ταξινομήσεων. Κατά την αξιολόγηση να ελέγξετε την επίδραση των εμπλεκομένων παραμέτρων, όπως ο αριθμός των οπτικών λέξεων (Βήμα 1), ο αριθμός των πλησιέστερων γειτόνων (Βήμα 3α) και ο τύπος του πυρήνα (kernel) του SVM (Βήμα 3β).

ΠΕΡΙΕΧΟΜΕΝΑ

1.Ανάλυση του μοντέλου Bag-of Visual Words

1.1 Γενικά για το μοντέλο

Στην παρούσα εργασία θέλουμε να ταξινομήσουμε αυτόματα άγνωστες εικόνες σε κλάσεις που έχουμε εκπαιδεύσει με βάση το περιεχόμενο των εικόνων. Το μοντέλο που θα χρησιμοποιήσουμε για το σκοπό αυτό ονομάζεται Bag of Visual Words(BOW).

Στο μοντέλο αυτό, ουσιαστικά «γκρουπάρουμε» τα keypoints των εικόνων σαν «λέξεις» που περιγράφουν την εικόνα.

Πιο συγκεκριμένα, Το μοντέλο Bag of Visual Words έχει ως στόχο την κωδικοποίηση του συνόλου των τοπικών χαρακτηριστικών που εντοπίστηκαν σε μια εικόνα, σε μια καθολική αναπαράσταση της εικόνας

Το μοντέλο BOVW περιλαμβάνει τρία διακριτά βήματα:

1. Τον υπολογισμό ενός λεξικού χαρακτηριστικών από τις εικόνες εκπαίδευσης.

2. Την κωδικοποίηση των εικόνων εκπαίδευσης με τη χρήση του λεξικού

3. Την κωδικοποίηση της εικόνας ερωτήματος (άγνωστη)[1]

1.2Δημιουργία λεξικού

Για τον υπολογισμό του λεξικού χαρακτηριστικών πρέπει ουσιαστικά να εξάγουμε όλα τα σημαντικά σημεία (keypoints όπως είδαμε στην προηγουμένη εργασία) στη συνέχεια τις ομαδοποιούμε(Clustering) με την χρήση του αλγορίθμου K-means.

1.2.1 K-means Algorithm

Ο αλγόριθμος K-means clustering ομαδοποιεί τα χαρακτηριστικά με τον παρακάτω τρόπο :

Έστω ότι έχουμε X={x1,x2,x3…xn} χαρακτηριστικά και V={v1,v2,v3,…,vc} τα κέντρα των διαφορετικών ομάδων. Όπου n ο συνολικός αριθμός των χαρακτηριστικών από τις εικόνες που έχουμε ενώ c πόσες «ομάδες» θέλουμε να κάνουμε (το ορίζουμε εμείς).

Τότε τα βήματα του αλγορίθμου είναι :

1)διάλεξε τυχαία c κέντρα

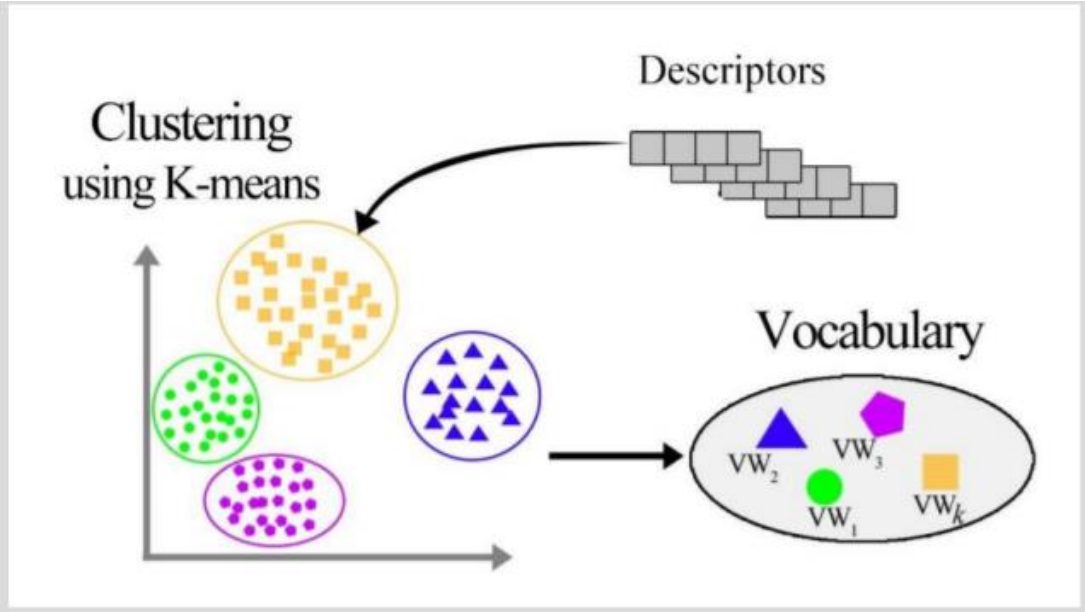
2)υπολόγισε για όλα τα σημεία την απόσταση τους από όλα τα κέντρα

3)Ανάθεσε το κάθε χαρακτηριστικά στην ομάδα με κέντρο που έχει την μικρότερη απόσταση

4)υπολόγισε το νέο κέντρο για κάθε ομάδα ως τον μέσο όρο των χαρακτηριστικών που έχει η ομάδα αυτή

5)υπολόγισε την απόσταση μεταξύ κάθε χαρακτηριστικού και του νέου κέντρου

6)Αν κανένα σημείο δεν έχει κέντρο διαφορετικής ομάδας πλέον πιο κοντινό σταμάτα αλλιώς επανέλαβε από το βήμα 3.



Εικόνα 1- Αναπαράσταση του K-means στα χαρακτηριστικά για την δημιουργία του λεξικού[1]

1.3.Κωδικοποίηση(Encoding)

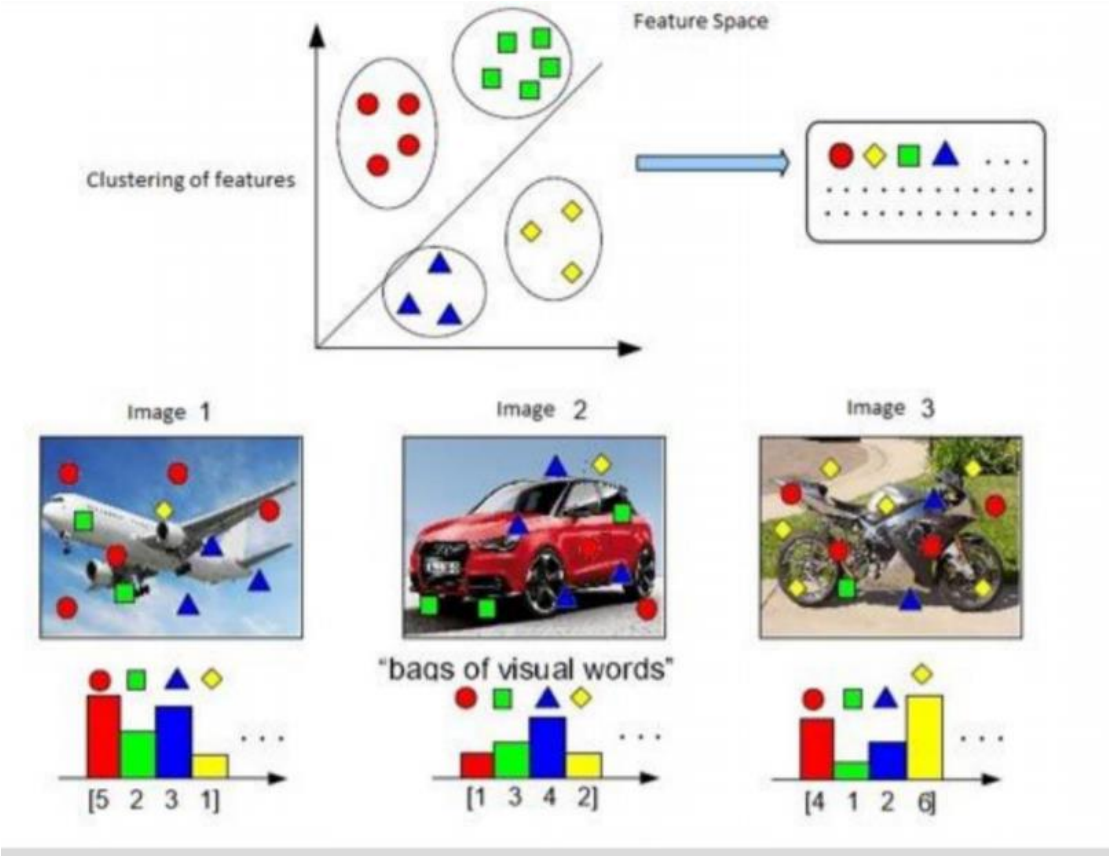
Η κωδικοποίηση της εικόνας γίνεται μετά τη δημιουργία του λεξικού χαρακτηριστικών. Για την κωδικοποίηση μιας εικόνας ακολουθούνται τα εξής βήματα:

1. Εξαγωγή των τοπικών χαρακτηριστικών

2. Συσχέτιση (matching) κάθε τοπικού χαρακτηριστικού με την κοντινότερη λέξη του λεξικού με απόσταση L2 ή L1

3. Δημιουργία ιστογράμματος με τη συχνότητα εμφάνισης κάθε λέξης μέσα στην εικόνα

Αυτήν την κωδικοποίηση την κάνουμε τόσο για τις εικόνες εκπαίδευσης όσο και για τις εικόνες που θέλουμε να ταξινομήσουμε.



Εικόνα 2 Οπτικοποίηση της Κωδικοποίησης[1]

1.4 Ταξινόμηση

Έχοντας κωδικοποιήσει και την εικόνα που θέλουμε να κωδικοποιήσουμε μπορούμε να την ταξινομήσουμε σε κάποια από τις κλάσεις μας. Για να το κάνουμε μπορούμε στο πλαίσιο της παρούσας εργασίας να χρησιμοποιήσουμε τους εξής αλγορίθμους:

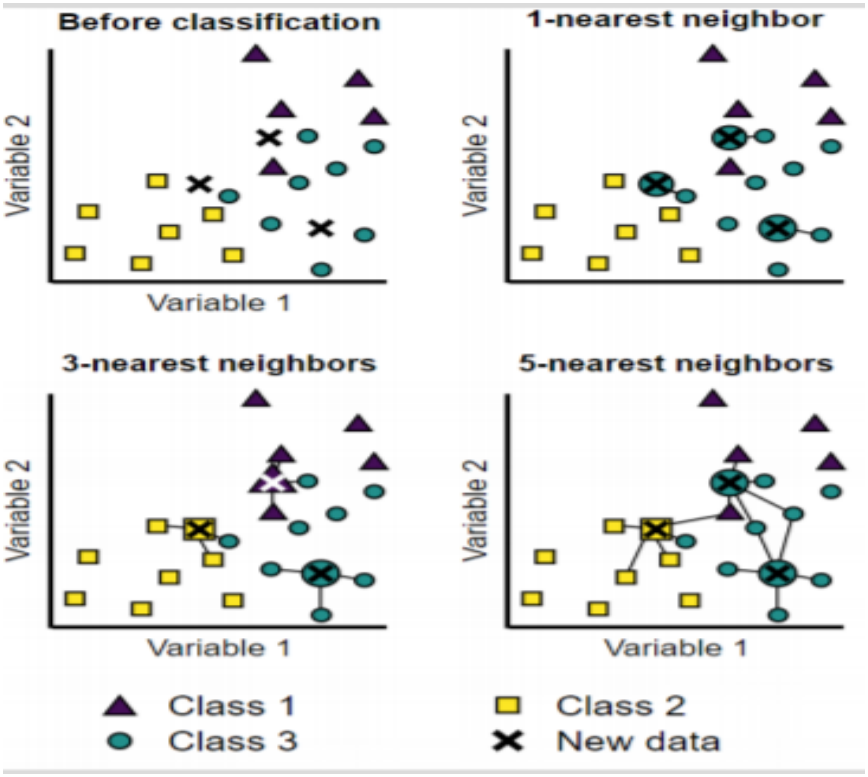
A) K-Nearest Neighbors

B) Support Vector Machine

1.4.1 K-Nearest Neighbors

Γενικά ο συγκεκριμένος αλγόριθμος λειτουργεί ως εξής:

Για κάθε άγνωστο σημείο, ελέγχονται οι K κοντινότεροι γείτονες του. Η κλάση που ανατίθεται στο άγνωστο σημείο εξάγεται από την πλειοψηφία των γειτόνων. Έχει το πλεονέκτημα ότι υποστηρίζει περισσότερες των 2 κλάσεων.

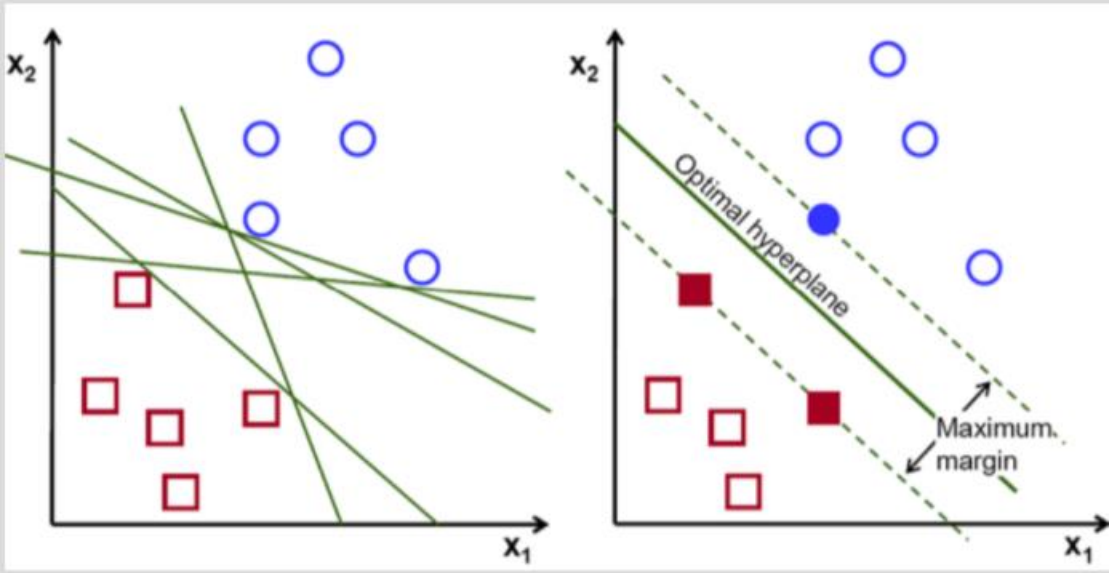


Εικόνα 2 - K-nearest neighbor[2]

Στη δικιά μας περίπτωση «σημεία» αποτελούν τα ιστογράμματα που έχουμε υπολογίσει για κάθε εικόνα όπως περιγράψαμε παραπάνω, και ως απόσταση ορίζουμε την L1 ή L2 απόσταση των σημείων αυτών.

1.4.2 Support Vector Machines

Υπολογίζει ένα υπερ-επίπεδο που διαχωρίζει βέλτιστα 2 κλάσεις. Το διαχωριστικό υπερ-επίπεδο θα πρέπει να μεγιστοποιεί την απόσταση μεταξύ των κλάσεων. Να σημειώσουμε ότι υποστηρίζει μόνο προβλήματα 2 κλάσεων (δυαδικά – binary).



Εικόνα 4 - SVM παράδειγμα [2]

Σε περίπτωση που τα δεδομένα είναι μη γραμμικώς διαχωριζόμενα τότε χρησιμοποιούμε ένα Svm Kernel Trick. Τα δεδομένα προβάλλονται σε έναν χώρο περισσότερων διαστάσεων (προσθέτοντας χαρακτηριστικά) .Ο γραμμικός διαχωρισμός στον επαυξημένο χώρο αντιστοιχεί σε μη γραμμικό διαχωρισμό στον αρχικό.

Στα πλαίσια της εργασίας θα χρησιμοποιήσουμε ένα SVM για κάθε κλάση (One vs All). Όπου ουσιαστικά ο διαχωρισμός γίνεται για κάθε SVM κλάσης μεταξύ των δεδομένων που ανήκουν στην κλάση αυτή και σε όλα τα υπόλοιπα. Στη συνέχεια για κάθε εικόνα που θέλουμε να ταξινομήσουμε συγκρίνουμε τα αποτελέσματα με κάθε SVM και διαλέγουμε την καλύτερη πρόβλεψη

2. Ανάλυση του κώδικα υλοποίησης μου

2.1 Γενική Περιγραφή

Η υλοποίση μου περιλαμβάνει 4 αρχεία κώδικα, έτσι ώστε να μπορώ όταν αλλάζω διαφορετικές παραμέτρους του κώδικαή άμα αλλάξω το test dataset μου να μην χρειάζεται να υπολογίζω πράγματα από την αρχή .

Τα αρχεία είναι τα εξής:

1) **CreateVoc_CreateTrainDiscs.py**
   Στο αρχείο αυτό δημιουργόυμε του λεξικό (Vocabulary) που περιγράψαμε παρπάνω καθώς και κάνουμε το encoding των εικόνων εκπαίδευσης (Δημιουργία Histograms)
   Τέλος δημιουργώ ένα αρχείο με ετικέτες για κάθε εικόνα εκπαίδευσης για να ξέρω σε ποια κλάση ανήκει η κάθε εικόνα εκπαίδευσης

2) **CreateTestDisc_Labels.py**
   Στο αρχείο αυτό κωδικοποιούμε και αποθηκεύουμε τα ιστογράμματα των τεστ εικόνων.
   Επίσης αποθηκεύω έναν πίνακα που έχουμε μέσα μια ετικέτα που δείχνει σε ποια κλάση ανήκει κανονικά η εικόνα αυτή**. Τον πίνακα αυτό τον δημιουργώ μόνο για να μπορέσω να ελέγχω το accuracy των predictions που έκανα και δεν έχει πρόσβαση σε αυτόν κανένας αλγόριθμος ταξινόμησης που έχω κάνει**

3) **trainSVM.py**
   Στο αρχείο αυτό εκπαιδεύω και αποθηκεύω τα Support Vector Machines μου(6 όσα είναι οι κλάσεις μου.)

4) **ClassifyAndEvaluate.py**
   Στο αρχείο αυτό κάνω την ταξινόμηση των τεστ εικόνων μου πρώτα με την υλοποίηση του knn και μετά με τα SVM που εκπαίδευσα παραπάνω. Συγκρίνω για κάθε εικόνα την η\ κλάση που της ανέθεσα στην ταξινόμηση σε σύγκριση με την πραγματική της και υπολογίζω την συνολική ακρίβεια της κάθε υλοποιήσης.

2.2. Αναλυτική ανάλυση κώδικα

Στο αρχείο **CreateVoc_CreateTrainDiscs.py** πιο συγκεκριμένα αρχικά φορτώνουμε τον φάκελο με τους φακέλους των εικόνων εκπαιδυεσης και δημιοτυργούμε τον περιγραφέα μας. Στην συνέχεια χρησιμοποιώντας την συνάρτηση που γράψαμε στο εργαστήριο extract_local_features θα εξάγουμε τους περιγραφείς των keypoints κάθε εικόνας και θα τους αποθηκεύσουμε στον πίνακα train_descs

```
imagedb_train = ('imagedb_train')

sift = cv.xfeatures2d_SIFT.create()
folders = os.listdir(imagedb_train)

def extract_local_features(path):
    img = cv.imread(path)

    kp = sift.detect(img)
    desc = sift.compute(img, kp)
    desc = desc[1]
    return desc

train_descs = np.zeros((0, 128), dtype=np.float32)
for folder in folders:
    current_folder = os.path.join(imagedb_train, folder)
    print(current_folder)  #gia na vlepw se poio simio ine o kodikas
    files = os.listdir(current_folder)
    for file in files:
        current_file = os.path.join(current_folder, file)
        desc = extract_local_features(current_file)
        train_descs = np.concatenate((train_descs, desc), axis=0)
```

Έχοντας πλέον όλα αυτά θα χρησιμοποιήσουμε τις συναρτήσεις της opencv για να εκτελέσουμε τον k-means και να δημιουργήσουμε το vocaulary μας και να το αποθηκεύσουμε στο αρχείο **vocabulary.npy**

```
term_crit = (cv.TERM_CRITERIA_EPS, 30, 0.1)
loss, assignments, vocabulary = cv.kmeans(train_descs.astype(np.float32),
50, None, term_crit, 1, 0)
np.save('vocabulary.npy', vocabulary)
```

όπου μερικά σημαντικά ορίσματα που δίνουμε στη συνάρτηση είναι τα δεδομένα που θέλουμε να κάνουμε cluster , στην περίπτωση μας τον κάθε περιγραφέα κάθε keypoint,Τον αριθμό των Clusters μας (δηλαδή των αριθμό των «λέξεων» του vocabulary μας καθώς και τα κριτήρια τερμτισμού δηλαδή την επιθυμητή ακρίβεια και τον μέγιστο αριθμό επαναλήψεων που θέλουμε.

Στη συνέχεια στο ίδιο αρχείο θέλουμε να δημιουργήσομε τα ιστογράμματα των εικόνων μας να κωδικοποιήσουμε δηλαδή τις εικόνες εκπαίδευσης μας. Αυτό το κάνω ορίζοντας την συνάρτηση :

```
def getBovwDescriptor(desc, vocabulary):
    bow_desc = np.zeros((1, vocabulary.shape[0]), dtype=np.float32)
    for d in range(desc.shape[0]):
        distances = desc[d, :] - vocabulary
        distances = np.abs(distances)
        distances = np.sum(distances, axis=1)
        mini = np.argmin(distances)
        bow_desc[0, mini] += 1
    return bow_desc
```

όπου σαν παραμέτρους εισάγω όλους τους περιγραφείς μιας εικόνας καθώς και το λεξικό μου. Η συνάρτηση αυτή για κάθε περιγραφέα υπολογίζει την L1 απόσταση του περιγραφέα με όλες τις λέξεις του λεξικού και στην ελάχιστη απόσταση προσθέτει συν 1 σε ένα πίνακα που θα δείχνει την συχνότητα εμφάνισης αυτής της λέξης. Αυτόν τον πίνακα επιστρέφει στο τέλος. Εδώ να πω ότι δεν έκανα κάποια κανονικοποίηση των ιστογραμμάτων μου καθώς δεν μου βελτίωνε σχεδόν καθόλου τα αποτελέσματα αλλά έκανε πιο αργό το πρόγραμμα μου.

Οπότε εδώ

```
bow_descs = np.zeros((0, vocabulary.shape[0]), dtype=np.float32)
for folder in folders:
    current_folder = os.path.join(imagedb_train, folder)
    print(current_folder)
    files = os.listdir(current_folder)
    for file in files:
        current_file = os.path.join(current_folder, file)
        print(current_file)
        desc = extract_local_features(current_file)
        bow_desc = getBovwDescriptor(desc, vocabulary)
        bow_descs = np.concatenate((bow_descs, bow_desc), axis=0)

train_class_labels=np.concatenate((train_class_labels,current_label),
axis=0)
```

```
    current_label[0] = current_label[0] + 1

np.save('bow_descs.npy', bow_descs)
np.save('train_class_labels.npy', train_class_labels)
```

υπολογίζω όλα τα ιστογράμματα και τα αποθηκεύω στον πίνακα bow_descs.npy **ενώ ταυτόχρονα φτιάχνω έναν πίνακα που ουσιαστικά η κάθε θέση του αντιστοιχεί σε κάθε εικόνα που έχω και η τιμή της κάθε θέσης του πίνακα μας δείχνει σε ποια κλάση ανήκει αυτή η εικόνα κάτι χρήσιμο για τον έλεγχο αργότερα των τεστ εικόνων και τον αποθηκεύω και αυτόν τον πίνακα ως train_class_labels.npy**

Στη συνέχεια κάνουμε ακριβώς το ίδιο για τις τεστ εικόνες μας με ακριβώς την ίδια διαδικασία με μόνη αλλαγή τα ονόματα των αρχείων που αποθηκεύουν έχουν το πρόθεμα test αντί για train.

Παιρνόντας στο classification θα κάνουμε ένα άλμα πρώτα στην εκπαίδευση του SVM καθώς γίνεται σε ξεχωριστό αρχείο ενώ μετά το classify με knn και SVM με τη σύγκριση τους γίνεται με ένα αρχείο στο τέλος.

Οπότε στο αρχείο train.SVM φορτώνουμε αρχικά τις κωδικοποιημένες εικόνες (Histogramms) καθώς και τις ετικέτες με τις κλάσεις τους.

```
training_data = np.load('bow_descs.npy')
training_class_labels = np.load('train_class_labels.npy')
```

Στη συνέχεια δημιουργούμε τα SVM με την παρακάτω διαδικασία:

```
for i in range (1,7):
    current_labels = np.zeros((training_class_labels.shape),dtype=int)
    n1 = training_class_labels.shape[0]
    for j in range (n1):
        if training_class_labels[j] == i:
            current_labels[j] = 1;
    svm = cv.ml.SVM_create()
    svm.setType(cv.ml.SVM_C_SVC)
    svm.setKernel(cv.ml.SVM_RBF)
    svm.setTermCriteria((cv.TERM_CRITERIA_MAX_ITER, 100, 1e-6))
    svm.trainAuto(training_data.astype(np.float32), cv.ml.ROW_SAMPLE,
current_labels)
    SVM_file_name = "SVM" + str(i)
    svm.save(SVM_file_name)
    print (SVM_file_name)
```

Εδώ ουσιαστικά για κάθε κλαση (1 μεχρι και 6) κάνουμε έναν πίνακα για να βάζουμε κάθε φορά την κάθε εικόνα αν ανήκει στην αντίστοιχη κλάση ή όχι (1 ή 0) κάτι που αποτελεί παράμετρο για κάθε SVM που θέλουμε να φτιάξουμε. Εδώ χρησιμοποιηούμε την συνάρτηση autTrain παραμετροποιώντας όπως στο

εργαστήριο διαλέγοντας και το κερνελ που θέλουμε να χρησιμοποιήσουμε. Στη συνέχεια το κάθε SVM το αποθηκεύουμε με όνομα ανάλογα την κλάση που αντιστοιχεί στο one vs all.

Τέλος έχοντας κωδικοποιήσει τα πάντα και έχοντας εκπαιδεύσει τα SVM μας μπορούμε να περάσουμε στο classification και evaluation στο τελευταίο αρχείο δηλαδή όπου ξεκινάμε με τον α) K-nn

Όπου αρχικά φορτώνουμε αυτά από πριν που θα χρειαστουμε και στη συνέχεια ,

Υπολογίζουμε την L1 απόσταση του ιστογράμματος της εικόνας που θέλουμε να κάνουμε classify με όλα τα ιστογράμματα των εικόνων εκπαίδευσης και κάνουμε sort από την μικρότερη στην μεγαλύτερη. Τις θέσεις των αποστάσεων δηλαδή το νούμερο της εικόνας. Έπειτα κοιτάμε τις πρώτες κ εικόνες σε ποια κλάση ανήκουν αυξάνοντας έναν δείκτη για κάθε κλαση σε έναν πίνακα (sum_labels) οπότε αυτός ο πίνακας μας δείχνει σε ποια κλάση ανήκουν οι πιο κοντινές εικόνες στην εικόνα μας ποσοτικά. Άρα παίρνοντας την θέση με τη μεγαλύτερη τιμή (**προσθέτοντας και 1 γιατι οι θέσεις του πίνακα ξεκινάνε από 0 ενώ οι κλάσεις από 1**) έχουμε το classification της εικόνας μας. Μπορούμε τώρα να δούμε για την τεστ εικόνα αυτή ποια είναι η πραγματική της κλάση (κάνοντας και print τα δυο στοιχεία αυτά) στον πίνακα που φτιάξαμε πριν με τις ετικέτες και αν είναι ίδια να αυξάνουμε ένα counter

```
vocabulary = np.load('vocabulary.npy')
test_bow_descs = np.load('test_bow_descs.npy')
test_class_labels = np.load('test_class_labels.npy')

#K-Nearest Neighbors

training_data = np.load('bow_descs.npy')
training_class_labels = np.load('train_class_labels.npy')
K = 3
counter=0
for i in range(test_class_labels.shape[0]):
    sum_labels = np.zeros(6, dtype=int)
    distances = test_bow_descs[i] - training_data
    distances = np.abs(distances)
    distances = np.sum(distances, axis=1)
    sorted_ids = np.argsort(distances)
    for j in range (K):
        sum_labels[training_class_labels[sorted_ids[j]]-1]+= 1
    max_label = np.argmax(sum_labels, axis=0)
    print("Predictied Class(K-nn) : ",max_label+1," Real Class:
",test_class_labels[i])
    if max_label+1 == test_class_labels[i]:
        counter+=1

precision = float(counter)/float(test_class_labels.shape[0])
print("The number of Correct  Matches Using K-nn is ",counter,"out of
",test_class_labels.shape[0]," pictures")
print("The accuracy of the K-nn model is {a:.5f}%".format(a=precision*100))
```

Έτσι μπορούμε μετά να υπολογ΄σουμε και το precision και να το τυπώσουμε

Τώρα για τον SVM φτιάχνουμε μια συνάρτηση την :

```
def Image_Classif_SVM(bow_d, svm_list):
    min_prediction= 9999999999999999999
    min_svm= -1
    bow_d = np.expand_dims(bow_d, axis=1)
    bow_d = np.transpose(bow_d)
    for svm in svm_list:
        prediction = svm.predict(bow_d.astype(np.float32),
flags=cv.ml.STAT_MODEL_RAW_OUTPUT)[1]
        if prediction[0] <= min_prediction:
            min_prediction = prediction[0]
            min_svm=svm_list.index(svm)+1
    return min_svm
```

όπου σαν όρισμα βάζω την κωδικοποίηση της εικόνας(ιστόγραμμα) και την λίστα των svm μου. Εδώ για κάθε svm κάνω ένα prediction για το που ανήκει η εικόνα και στο τέλος κρατάω και επιστρέφω τον δείκτη το SVM που έκανε την καλύτερη πρόβλεψη (μικρότερο νούμερο) ώστε να ανήκει σε αυτή την κλάση η εικόνα που θέλω να ταξινομήσω

Άρα για κάθε εικόνα κάνω το εξής

```
counter=0
for i in range (test_bow_descs.shape[0]):
    classif_label=Image_Classif_SVM(test_bow_descs[i], list_of_SVMs)
    print("Predictied Class(SVM) : ",str(classif_label)," Real Class:
",str(test_class_labels[i]))
    if classif_label == test_class_labels[i] :
        counter = counter+1

precision = float(counter)/float(test_class_labels.shape[0])
print("The number of Correct  Matches Using SVM is ",counter,"out of
",test_class_labels.shape[0]," pictures")
print("The accuracy of the SVM model is {a:.5f}%".format(a=precision*100))
```

Φορτώνω τα svm μου καλώ την συνάρτηση για κάθε εικόνα τεστ και κάνω την ίδια σύγκριση με πριν για να ελέγχω το αποτέλεσμα με μόνη διαφορά ότι πλέον επειδή η συνάρτηση επιστρέφει με σωστή αρίθμηση το label δεν χρειάζεται να γίνει +1

Πλέον μπορούμε να δούμε στην κονσόλα ένα παράδειγμα αποτελέσματος:

```
Predictied Class(K-nn) :  3  Real Class:  [1]
Predictied Class(K-nn) :  5  Real Class:  [1]
Predictied Class(K-nn) :  5  Real Class:  [1]
Predictied Class(K-nn) :  1  Real Class:  [1]
Predictied Class(K-nn) :  2  Real Class:  [1]
Predictied Class(K-nn) :  1  Real Class:  [1]
Predictied Class(K-nn) :  6  Real Class:  [1]
Predictied Class(K-nn) :  5  Real Class:  [1]
Predictied Class(K-nn) :  1  Real Class:  [1]
Predictied Class(K-nn) :  6  Real Class:  [1]
Predictied Class(K-nn) :  2  Real Class:  [2]
Predictied Class(K-nn) :  2  Real Class:  [2]
Predictied Class(K-nn) :  2  Real Class:  [2]
Predictied Class(K-nn) :  2  Real Class:  [2]
Predictied Class(K-nn) :  2  Real Class:  [2]
Predictied Class(K-nn) :  2  Real Class:  [2]
Predictied Class(K-nn) :  2  Real Class:  [2]
Predictied Class(K-nn) :  2  Real Class:  [2]
Predictied Class(K-nn) :  5  Real Class:  [2]
Predictied Class(K-nn) :  2  Real Class:  [2]
Predictied Class(K-nn) :  3  Real Class:  [3]
Predictied Class(K-nn) :  1  Real Class:  [3]
Predictied Class(K-nn) :  3  Real Class:  [3]
Predictied Class(K-nn) :  2  Real Class:  [3]
Predictied Class(K-nn) :  3  Real Class:  [3]
Predictied Class(K-nn) :  4  Real Class:  [3]
Predictied Class(K-nn) :  2  Real Class:  [3]
Predictied Class(K-nn) :  2  Real Class:  [3]
Predictied Class(K-nn) :  1  Real Class:  [3]
Predictied Class(K-nn) :  4  Real Class:  [4]
Predictied Class(K-nn) :  4  Real Class:  [4]
Predictied Class(K-nn) :  4  Real Class:  [4]
```

```
Predictied Class(K-nn) :  4  Real Class:  [4]
Predictied Class(K-nn) :  4  Real Class:  [4]
Predictied Class(K-nn) :  4  Real Class:  [4]
Predictied Class(K-nn) :  4  Real Class:  [4]
Predictied Class(K-nn) :  4  Real Class:  [4]
Predictied Class(K-nn) :  3  Real Class:  [4]
Predictied Class(K-nn) :  4  Real Class:  [4]
Predictied Class(K-nn) :  1  Real Class:  [5]
Predictied Class(K-nn) :  5  Real Class:  [5]
Predictied Class(K-nn) :  5  Real Class:  [5]
Predictied Class(K-nn) :  5  Real Class:  [5]
Predictied Class(K-nn) :  5  Real Class:  [5]
Predictied Class(K-nn) :  5  Real Class:  [5]
Predictied Class(K-nn) :  5  Real Class:  [5]
Predictied Class(K-nn) :  5  Real Class:  [5]
Predictied Class(K-nn) :  5  Real Class:  [5]
Predictied Class(K-nn) :  5  Real Class:  [5]
Predictied Class(K-nn) :  6  Real Class:  [6]
Predictied Class(K-nn) :  6  Real Class:  [6]
Predictied Class(K-nn) :  6  Real Class:  [6]
Predictied Class(K-nn) :  6  Real Class:  [6]
Predictied Class(K-nn) :  6  Real Class:  [6]
Predictied Class(K-nn) :  6  Real Class:  [6]
Predictied Class(K-nn) :  6  Real Class:  [6]
Predictied Class(K-nn) :  5  Real Class:  [6]
Predictied Class(K-nn) :  5  Real Class:  [6]
Predictied Class(K-nn) :  1  Real Class:  [6]
Predictied Class(K-nn) :  1  Real Class:  [6]
The number of Correct  Matches Using K-nn is  42 out of  62  pictures
The accuracy of the K-nn model is 67.74194%
```

```
Predictied Class(SVM) :  4  Real Class:  [1]
Predictied Class(SVM) :  5  Real Class:  [1]
Predictied Class(SVM) :  5  Real Class:  [1]
Predictied Class(SVM) :  5  Real Class:  [1]
Predictied Class(SVM) :  2  Real Class:  [1]
Predictied Class(SVM) :  1  Real Class:  [1]
Predictied Class(SVM) :  6  Real Class:  [1]
Predictied Class(SVM) :  5  Real Class:  [1]
Predictied Class(SVM) :  5  Real Class:  [1]
Predictied Class(SVM) :  6  Real Class:  [1]
Predictied Class(SVM) :  2  Real Class:  [2]
Predictied Class(SVM) :  2  Real Class:  [2]
Predictied Class(SVM) :  2  Real Class:  [2]
Predictied Class(SVM) :  2  Real Class:  [2]
Predictied Class(SVM) :  2  Real Class:  [2]
Predictied Class(SVM) :  2  Real Class:  [2]
Predictied Class(SVM) :  2  Real Class:  [2]
Predictied Class(SVM) :  3  Real Class:  [2]
Predictied Class(SVM) :  5  Real Class:  [2]
Predictied Class(SVM) :  2  Real Class:  [2]
Predictied Class(SVM) :  3  Real Class:  [3]
Predictied Class(SVM) :  2  Real Class:  [3]
Predictied Class(SVM) :  3  Real Class:  [3]
Predictied Class(SVM) :  2  Real Class:  [3]
Predictied Class(SVM) :  3  Real Class:  [3]
Predictied Class(SVM) :  4  Real Class:  [3]
Predictied Class(SVM) :  2  Real Class:  [3]
Predictied Class(SVM) :  2  Real Class:  [3]
Predictied Class(SVM) :  3  Real Class:  [3]
Predictied Class(SVM) :  4  Real Class:  [4]
Predictied Class(SVM) :  4  Real Class:  [4]
Predictied Class(SVM) :  4  Real Class:  [4]
```

```
Predictied Class(SVM) :  4  Real Class:  [4]
Predictied Class(SVM) :  4  Real Class:  [4]
Predictied Class(SVM) :  4  Real Class:  [4]
Predictied Class(SVM) :  4  Real Class:  [4]
Predictied Class(SVM) :  4  Real Class:  [4]
Predictied Class(SVM) :  4  Real Class:  [4]
Predictied Class(SVM) :  4  Real Class:  [4]
Predictied Class(SVM) :  4  Real Class:  [4]
Predictied Class(SVM) :  4  Real Class:  [4]
Predictied Class(SVM) :  5  Real Class:  [5]
Predictied Class(SVM) :  5  Real Class:  [5]
Predictied Class(SVM) :  5  Real Class:  [5]
Predictied Class(SVM) :  5  Real Class:  [5]
Predictied Class(SVM) :  5  Real Class:  [5]
Predictied Class(SVM) :  5  Real Class:  [5]
Predictied Class(SVM) :  5  Real Class:  [5]
Predictied Class(SVM) :  5  Real Class:  [5]
Predictied Class(SVM) :  5  Real Class:  [5]
Predictied Class(SVM) :  5  Real Class:  [5]
Predictied Class(SVM) :  5  Real Class:  [5]
Predictied Class(SVM) :  5  Real Class:  [5]
Predictied Class(SVM) :  6  Real Class:  [6]
Predictied Class(SVM) :  6  Real Class:  [6]
Predictied Class(SVM) :  6  Real Class:  [6]
Predictied Class(SVM) :  6  Real Class:  [6]
Predictied Class(SVM) :  6  Real Class:  [6]
Predictied Class(SVM) :  6  Real Class:  [6]
Predictied Class(SVM) :  6  Real Class:  [6]
Predictied Class(SVM) :  6  Real Class:  [6]
Predictied Class(SVM) :  6  Real Class:  [6]
Predictied Class(SVM) :  6  Real Class:  [6]
Predictied Class(SVM) :  5  Real Class:  [6]
Predictied Class(SVM) :  6  Real Class:  [6]
The number of Correct  Matches Using SVM is  45 out of  62  pictures
The accuracy of the SVM model is 72.58065%
```

3. Αξιολόγηση του συστήματος

Έχοντας δείξει πως θα κάνουμε τα μετρικά μας για την αξιολόγηση του συστήματος θα δούμε σε διάφορες παραμέτρους (κρατώντας άλλες σταθερές) πως επηρεάζεται η ακρίβεια του

Α) Κρατώντας αχικά σταθερό το knn στο 3 και το κερνελ του SVM στο RBF (επειδή είναι η default value) δηλαδή το κέρνελ Radial basis function (RBF) Θα αλλάξουμε τον αριθμό των λέξεων και θα δούμε το αποτέλεσμα

Words	SVM(%)	K-nn(%)
20	66,12903	59,67742
50	72,58065	67,74194
100	70,96774	70,96774
200	83,87097	67,74194
500	69,35484	45,16129

Παρτατηρώ ότι μέχρι ένα σημείο αυξάνεται η ακρίβεια και μετά ξαναπέφτει, για SVM το βέλτιστο ήταν 200 για knn ήταν 100

Β)Κρατώντας τώρα σταθερές τις λέξεις στο 100 θα δούμε πως αλλάζοντας το K έχουμε αλλαγή στην ακρίβεια του K-nn

K	Accuracy
1	64,51613
2	61,29032
3	70,96774
5	69,35484
7	69,35484
10	70,96774
20	62,90323

Οι διακυμάνσεις εδώ είναι σχετικά μικρές

Γ)Τώρα κρατώντας σταθερές τις λέξεις στα 100 θα δούμε τα διαφορετικά kernel types τι αντίκτυπο έχουν στο accuracy.

Kernel Type	Accuracy(%)
Radial basis function (RBF)	70,96774
Linear Kernel	38,70968
SIGMOID	17,74194
Expotential Chi2 Kernel	77,41935
Histogram intersection kernel	46,77419

Το καλύτερο αποτέλεσμα το είχε το Sigmoid Kernel και εδώ είχαμε μεγάλες αποκλίσεις.

4.ΠΗΓΕΣ-ΑΝΑΦΟΡΕΣ

[1]. Διαφάνειες εργαστηρίου, Μάθημα 8 – Bag of Visual Words

[2]. Διαφάνειες εργαστηρίου, Μάθημα 9 – Ταξινόμηση