






+

ΕΡΓΑΣΙΑ 3**ΟΡΑΣΗ ΥΠΟΛΟΓΙΣΤΩΝ****ΟΝΟΜΑΤΕΠΩΝΥΜΟ: Ζαχάρη Βάια****ΑΜ: 58161****ΕΙΣΑΓΩΓΗ:**

Στη συγκεκριμένη εργασία κληθήκαμε να υλοποιήσουμε ένα πρόγραμμα για ταξινόμηση πολλαπλών κλάσεων. Έχοντας στη διάθεση μας ένα dataset εικόνων θα προσπαθήσουμε να εξάγουμε ένα λεξικό με τα αντιπροσωπευτικά χαρακτηριστικά αυτών. Στη συνέχεια, αν εισάγουμε νέες εικόνες θα πρέπει αυτές να ενταχθούν σε μία από τις πιθανές κλάσεις με βάση τα δικά τους χαρακτηριστικά και μέσω δύο αλγορίθμων ταξινόμησης, τον KNearestNeighbors (knn) και τον System Vector Machine(svm). Επιπλέον θα εξαχθούν συμπεράσματα για τα αποτελέσματα και την ακρίβεια απόδοσης της νέας εικόνας στη σωστή κλάση.

ΜΕΘΟΔΟΛΟΓΙΑ:

-  Δημιουργία λεξικού (vocabulary.npy)
-  Δημιουργία του συνόλου εικόνων με τα χαρακτηριστικά τους (index.npy)
-  Testing εικόνων από το dataset imaged_test με τον manual knn αλγόριθμο.
-  Training του svm αλγόριθμου για τις πέντε κλάσεις.
-  Testing του svm με βάση το dataset imaged_test.

ΒΗΜΑ 1^ο: ΛΕΞΙΚΟ

Όπως γνωρίζουμε ο υπολογιστής για να μπορέσει να διακρίνει συγκεκριμένα αντικείμενα μέσα σε μία εικόνα πρέπει να μπορεί να τα διαχωρίζει από τα υπόλοιπα με βάσει κάποιων χαρακτηριστικών τους. Έτσι δημιουργούνται οι κλάσεις, δηλαδή ομάδες αντικειμένων με ίδια χαρακτηριστικά. Το κάθε αντικείμενο μέσα στην εικόνα εκφράζεται από ένα «σακίδιο» λέξεων, το οποίο παρουσιάζει τοπικά χαρακτηριστικά. Με τον υπολογισμό των keypoints μπορούμε να βρούμε τα σημεία τοπικού ενδιαφέροντος και διανύσματα με χαρακτηριστικά για τη κλάση. Το σύνολο αυτών των τοπικών σημείων ενδιαφέροντος συνιστά το λεξικό. Η οπτική λέξη είναι τα χαρακτηριστικά του διανύσματος που υπολογίζονται στα σημεία ενδιαφέροντος.

Διαδικασία ταιριάσματος:

- Δημιουργώ το λεξικό με διάφορα σακίδια από εικόνες.
- Η νέα εικόνα χωρίζεται σε τοπικά χαρακτηριστικά.
- Δημιουργείται ένα ιστόγραμμα, όπου στον οριζόντιο άξονα περιέχει τα αντιπροσωπευτικά χαρακτηριστικά του λεξικού και στον κάθετο άξονα το πλήθος εμφάνισης κάθε χαρακτηριστικού στην νέα εικόνα και έτσι έχοντας ένα νέο σακίδιο προσπαθούμε να το ταυτίσουμε με ένα από τα ήδη υπάρχοντα. Δηλαδή να αποδώσουμε τη νέα εικόνα σε μία κλάση.

Αυτό το μοντέλο ταξινόμησης ονομάζεται «Bag of Visual Words» (BOWS) και όπως φάνηκε προσπαθεί να κωδικοποιήσει το σύνολο τοπικών χαρακτηριστικών που εντοπίστηκαν μέσα σε μία νέα εικόνα σε μια καθολική αναπαράσταση της εικόνας.

Ο κώδικας που δημιουργεί το λεξικό είναι στο αρχείο HW3_vocab_and_index1.py στο πρώτο μέρος. Όπως φαίνεται ενώνει τα χαρακτηριστικά που βρίσκει από τα αρχεία κάθε φακέλου με τις εικόνες εκπαίδευσης.

Στη συνέχεια, για να τα ομαδοποιήσει και να δημιουργήσει τις κλάσεις ή αλλιώς clusters, δίνονται τυχαία κέντρα τους με τη βοήθεια του K-Means αλγορίθμου.

```
trainer = cv.BOWKMeansTrainer(50, term_crit, 1, cv.KMEANS_PP_CENTERS)
vocabulary = trainer.cluster(train_descs.astype(np.float32))
```

Κάθε διάνυσμα χαρακτηριστικών είναι και ένα σημείο στο χώρο. Οπότε τα σημεία τα συσχετίζουμε με το κοντινότερο k cluster και άρα με μία κλάση.

Το «bag of words» αποθηκεύεται στο αρχείο «index.npy». Περιέχει μία λίστα με τις εικόνες εκπαίδευσης με τα χαρακτηριστικά τους.

Συναρτήσεις που υλοποιούν το μοντέλο «BOWS»:

```
descriptor_extractor = cv.BOWImgDescriptorExtractor(sift, cv.BFMatcher(cv.NORM_L2SQR))
descriptor_extractor.setVocabulary(vocabulary)
```

Εδώ υπολογίζονται τα τοπικά χαρακτηριστικά με βάση τα keypoints μιας εικόνας:

```
bow_desc = descriptor_extractor.compute(img, kp)
```

Τα συνολικά χαρακτηριστικά θα αποθηκευτούν στη λίστα bow_descs:

```
bow_descs = np.concatenate((bow_descs, bow_desc), axis=0)
```

και το index.npy:

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

Επιπλέον αποθηκεύουμε και το path από κάθε φάκελο εκπαίδευσης σε ένα αρχείο «paths.npy», γιατί θα μας φανεί χρήσιμο στη συνέχεια

ΒΗΜΑ 2^ο: ΤΑΞΙΝΟΜΗΣΗ ΚΑΙ ΤΕΣΤ ΜΕ knn(K-Nearest-Neighbors)

Η βασική ιδέα είναι να βρούμε ομοιότητα των χαρακτηριστικών της νέας εικόνας με αυτά των εκπαιδευμένων εικόνων, που βρίσκονται στο αρχείο index.npy. Όταν βρεθεί, τότε αποδίδουμε στη νέα εικόνα την κλάση με της οποίας τα χαρακτηριστικά ήταν πιο όμοια.

Αρχικά φορτώνουμε τα αρχεία index.npy και paths.npy. Με τη βοήθεια του paths.npy δημιουργούμε μία λίστα «labels» που θα παίρνει τιμές από το 1 έως 5 για τις πέντε κλάσεις αντίστοιχα. Ανάλογα με το path θα παίρνει και το αντίστοιχο νούμερο κλάσης. Να σημειωθεί ότι τα paths προήλθαν από τις εικόνες εκπαίδευσης. Τελικά θα πάρουμε μία λίστα με μέγεθος όσες οι εικόνες και θα φαίνεται σε μία σειρά πως είναι ταξινομημένες οι κλάσεις, ή αλλιώς τα διανύσματα χαρακτηριστικών αυτών.

Έπειτα εισάγουμε μία εικόνα και εξάγουμε τα τοπικά της χαρακτηριστικά. Για να μπορέσουμε να τα ομαδοποιήσουμε κατάλληλα βρίσκουμε πόσο όμοια είναι μεταξύ τους.

```
distances = np.sum((bow_desc - bow_descs) ** 2, axis=1)
```

Αφού υπολογίσουμε τις αποστάσεις, δηλαδή την ομοιότητα, τις ταξινομούμε με αύξουσα σειρά και παίρνουμε τις θέσεις αυτών.

```
retrieved_ids = np.argsort(distances)
```

Στο σημείο αυτό στο κώδικα μπορούμε να βρούμε τη πιο όμοια εικόνα με τη νέα εικόνα με τα paths των trained folders.

```
for id in retrieved_ids.tolist():
    result_img = cv.imread(img_paths[id])
    cv.imshow('results', result_img)
    cv.waitKey(0)
pass
```

Έτσι για παράδειγμα μπορώ να εισάγω τη παρακάτω εικόνα και να μου βγάλει όμοιες.

```
test_img = 'caltech/imagenet_test/145.motorbikes-101/145_0095.jpg'
```



Test image



1st Matched image



2st Matched image



7st Matched image

Παρατήρηση:

Όσο προχωρούσε η διαδικασία ταιριάσματος της εικόνας, αυτή που επιστρεφόταν δεν άνηκε στην ίδια κλάση. Όπως φαίνεται εδώ η αρχική εικόνα (test) είναι μία μοτοσυκλέτα, ενώ η 7^η που επιστράφηκε είναι ένα αεροπλάνο.

Σε συνέχεια με τα προηγούμενα, τώρα διαθέτουμε τη σειρά των σημείων με τις ελάχιστες αποστάσεις. Με βάση μία παράμετρο k , η οποία δείχνει πόσους κοντινούς γείτονες θέλουμε να λάβουμε υπόψιν μας, συγκρατάμε τις θέσεις από τις k πρώτες αποστάσεις. Με βάση αυτές τις θέσεις μέσα στη λίστα labels παίρνουμε το νούμερο της κλάσης. Αν αυτό γίνει για τους k γείτονες, θα ελέγχω ποια κλάση εμφανίζεται πιο συχνά και άρα αυτή θα είναι και η κλάση της εικόνας.

```
def most_frequent(List):
    return max(set(List), key=List.count)

for n in range(0,k):
    best_dist[n] = retrieved_ids[n]
    n = n + 1
```

Στη συνέχεια εκτυπώνει τη κλάση που προβλέπει ότι ανήκει η εικόνα.

Για να γίνει επαναληπτικά αυτό το testing για όλες τις εικόνες του imagenet_test, όλη η διαδικασία φαίνεται ως συνάρτηση testing(), με παράμετρο το path της εικόνας για το test. Η παραγωγή των paths για κάθε αρχείο και φάκελο γίνεται ως εξής:

```
for folder in test_folders: #παιρνω τα αρχεία καθε φακελου ..λίστα απο 97 στοιχει αμε τ
α ονοματα των φακελων
    files = os.listdir(folder) #φακελος / file
    for file in files:
        test_img = folder + '/' + file
```

```
cnt = testing(test_img)
total_tries = total_tries + 1
```

Για τη παραγωγή της ακρίβειας για την εύρεση της κλάσης τόσο συνολικά όσο και για ανα κλάση, πρέπει να υπολογίσουμε τον αριθμό επιτυχίας όλων των κλάσεων ως προς τον αριθμό όλων των προσπαθειών και τον αριθμό επιτυχίας της κάθε κλάσης ως προς τον αριθμό προσπαθειών της κάθε κλάσης. Έτσι:

- ✚ Δημιούργησα μία μεταβλητή (cnt) η οποία όταν εντός συνάρτησης επιλέγεται μία κλάση να δείχνει τον αριθμό της κλάσης και να την επιστρέφει η συνάρτηση.
- ✚ Όταν καλούμε τη συνάρτηση κρατάμε αυτό τον αριθμό και ελέγχουμε αν ο φάκελος που τεστάρουμε, κι άρα τη κλάση, συμπίπτει με τη κλάση που παράγει ο αλγόριθμος. Αν ναι αυξάνει το πλήθος επιτυχίας της κλάσης που πέτυχε.
- ✚ Σε ξεχωριστή if αυξάνουμε τις προσπάθειες της κάθε κλάσης ανάλογα με το path της εικόνας που τεστάρουμε.
- ✚ Τώρα μπορούμε να παράξουμε τα ποσοστά επιτυχίας της κάθε κλάσης και στο τέλος αθροίζουμε τις επιτυχίες της κάθε κλάσης και διαιρούμε με τις συνολικές προσπάθειες για αυτές, ώστε να πάρουμε το συνολικό ποσοστό επιτυχίας.

Παρακάτω βλέπουμε το αποτέλεσμα για $k = 11$.

```
Total_accuracy: 73.07692307692307 %
Accuracy_class_motorbikes: 40.0 %
Accuracy_class_school_bus: 77.77777777777779 %
Accuracy_class_touring_bike: 81.81818181818183 %
Accuracy_class_airplane: 100.0 %
Accuracy_class_car_side: 63.63636363636363 %
```

Γενικά προτιμούμε να επιλέγουμε το k , τον αριθμό των γειτόνων, ως έναν περιττό αριθμό, καθώς σε περίπτωση που είναι άρτιος υπάρχει περίπτωση όταν πρέπει να επιλέξει τη κλάση που συναντά πιο συχνά για τους γείτονες του, ο αριθμός των κλάσεων να είναι ο ίδιος για πάνω από μία κλάση και να διαλέγει τυχαία και άρα να μειώνεται η ακρίβεια.

Για να δοκιμάσω διάφορες τιμές του k και να δω τη συνολική ακρίβεια, έκανα μία επαναληπτική διαδικασία και τροποίησα τον κώδικα σε μία μεγάλη συνάρτηση, με όρισμα το k , όταν αυτό ικανοποιεί τη συνθήκη ότι είναι περιττός.

```
for n in range(5, 41):
    if (n % 2 != 0):
        accuracy=choosing_k(n)
        print('Total_accuracy: ', accuracy, '%')
        print('k=', n)
```

Ο κώδικας θα βρίσκεται ξεχωριστά στο αρχείο HW3_knn2_choose_k.py.

Παρακάτω φαίνονται τα αποτελέσματα:

Total_accuracy: 69.23076923076923 %	Total_accuracy: 73.07692307692307 %
k= 5	k= 33
Total_accuracy: 69.23076923076923 %	Total_accuracy: 69.23076923076923 %
k= 7	k= 35
Total_accuracy: 71.15384615384616 %	Total_accuracy: 67.3076923076923 %
k= 9	k= 37
Total_accuracy: 73.07692307692307 %	Total_accuracy: 71.15384615384616 %
k= 11	k= 39
Total_accuracy: 71.15384615384616 %	Total_accuracy: 71.15384615384616 %
k= 13	k= 41
Total_accuracy: 73.07692307692307 %	Total_accuracy: 71.15384615384616 %
k= 15	k= 43
Total_accuracy: 71.15384615384616 %	Total_accuracy: 65.38461538461539 %
k= 17	k= 45
Total_accuracy: 69.23076923076923 %	Total_accuracy: 67.3076923076923 %
k= 19	k= 47
Total_accuracy: 69.23076923076923 %	Total_accuracy: 65.38461538461539 %
k= 21	k= 49
Total_accuracy: 71.15384615384616 %	Total_accuracy: 67.3076923076923 %
k= 23	k= 51
Total_accuracy: 71.15384615384616 %	Total_accuracy: 67.3076923076923 %
k= 25	k= 53
Total_accuracy: 71.15384615384616 %	Total_accuracy: 65.38461538461539 %
k= 27	k= 55
Total_accuracy: 69.23076923076923 %	Total_accuracy: 67.3076923076923 %
k= 29	k= 57
Total_accuracy: 71.15384615384616 %	Total_accuracy: 65.38461538461539 %
k= 31	k= 59

ΠΑΡΑΤΗΡΗΣΗ:

Παρατηρούμε από τα αποτελέσματα, ότι το καλύτερο αποτέλεσμα ,το 73.076923% το βγάζει το k = 11, 15 και το 33. Καλύτερα βολεύει να κρατήσουμε το k = 11 , ώστε να μειωθεί και ο χρόνος υπολογισμού και να αυξηθεί η απόδοση.

ΒΗΜΑ 3^ο: TRAINING SVM-SYSTEM VECTOR MACHINE

Ο svm ταξινομητής προσπαθεί να λύσει το πρόβλημα με ακόμα πιο γεωμετρικό τρόπο. Προσπαθεί να διαχωρίσει τις κλάσεις μεταξύ τους με μία καμπύλη. Βέβαια επειδή τα σημεία δεν μπορούν να χωριστούν όλα με γραμμικό τρόπο, αναπαριστούμε αυτά τα σημεία σε ένα χώρο μεγαλύτερης διάστασης. Με τη βοήθεια του kernel γίνονται άμεσα οι υπολογισμοί για τις μετατροπές (transformations) και βρίσκει τρόπο να διαχωρίσει τα σημεία με βάση τις κλάσεις. Συγκεκριμένα, βρίσκει τις εξισώσεις του υπερεπίπεδου που θα διαχωρίσει τη μία κλάση από την άλλη. Αν ένα σημείο ανήκει στη κλάση θα φαίνεται από το αν βρίσκεται δεξιά ή αριστερά από το υπερεπίπεδο.

Επειδή όμως εδώ έχουμε πέντε κλάσεις θα δημιουργήσουμε πέντε υπερεπίπεδα και άρα χρειαζόμαστε πέντε svm. Για την εκπαίδευση του κάθε svm, θα υπολογίσουμε για κάθε κλάση ένα πίνακα labels, ο οποίος θα παίρνει τη τιμή 1 αν το σημείο βρίσκεται σε μία κλάση και τη τιμή 0 αν βρίσκεται σε κάποια από τις υπόλοιπες. Έτσι, αν καλέσουμε τη συνάρτηση trainAuto() και δώσουμε ως ένα από τα ορίσματα τους πίνακες αυτούς κάθε φορά, θα παραχθούν οι εξισώσεις των υπερεπιπέδων, πχ:

```
svm1.trainAuto(bow_descs, cv.ml.ROW_SAMPLE, labels1)
svm2.trainAuto(bow_descs, cv.ml.ROW_SAMPLE, labels2)
```

Στη συνέχεια αποθηκεύουμε αυτά τα svm.

ΒΗΜΑ 4^ο: TEST SVM

Σε αυτό το βήμα ξαναφορτώνουμε τα svm και μέσω της συνάρτησης svm.predict() εξάγουμε ένα νούμερο, το οποίο αν είναι αρνητικό σημαίνει ότι ανήκει στη κλάση που αντιστοιχεί το svm και η απόλυτη τιμή αυτού δίνει την απόσταση του σημείου από το υπερεπίπεδο του svm.

Πχ για τη κλάση motorbikes:

```
response1 = svm1.predict(bow_desc, flags=cv.ml.STAT_MODEL_RAW_OUTPUT)
```

Το bow_desc είναι τα χαρακτηριστικά της νέας εικόνας και τα υπολογίζουμε μέσω της συνάρτησης descriptor_extractor.compute()

```
bow_desc = descriptor_extractor.compute(img, kp)
```

Οπότε όταν τεστάρουμε μία εικόνα ελέγχουμε κάθε τιμή πρόβλεψης (response) και ανάλογα όποια είναι αρνητική, τότε σε εκείνης της κλάσης αντιστοιχεί η εικόνα. Αν για κάποιο λόγο βρεθούν να ανήκουν σε πάνω από μία κλάση, διαλέγουμε εκείνη που η απόλυτη τιμή της μεταβλητής response[1] έχει τη μεγαλύτερη τιμή. Εκείνης δηλαδή που η απόσταση από το υπερεπίπεδο είναι πιο μεγάλη, καθώς όσο πιο κοντά είναι ένα σημείο στο υπερεπίπεδο τόσο πιο οριακά ανήκει στη κλάση, ενώ το αντίθετο αντιστοιχεί σε ένα πιο confident σημείο της κλάσης. Επιπλέον αν όλα είναι θετικές τιμές, διαλέγουμε τη κλάση της μικρότερης απόστασης, αφού αυτή βρίσκεται πιο κοντά στο υπερεπίπεδο (πιο οριακά ανήκει στη κλάση που βρίσκεται) και άρα πιο κοντά στη κλάση που θα δώσει αρνητική τιμή.

Για να τεστάρουμε κάθε εικόνα και να υπολογίσουμε τα ποσοστά επιτυχίας, όπως προηγουμένως, θα υλοποιήσουμε την ίδια επαναληπτική διαδικασία. Τα αποτελέσματα παρουσιάζονται παρακάτω.

```
Total_accuracy: 80.76923076923077 %
Accuracy_class_motorbikes: 60.0 %
Accuracy_class_school_bus: 66.66666666666666 %
Accuracy_class_touring_bike: 90.9090909090909 %
Accuracy_class_airplane: 100.0 %
Accuracy_class_car_side: 81.81818181818183 %
```

ΣΥΜΠΕΡΑΣΜΑΤΑ

- ✚ Ιδιαίτερη αποτυχία παρουσίαζε η κλάση «motorbikes»(ερώτημα 5). Αυτό πιθανόν να οφειλέται γιατί εξωτερικά παραπέμπει και σε άλλα οχήματα με δύο ρόδες, όπως ποδήλατο. Επιπλέον δεν έχει ένα συγκεκριμένο χρώμα για να παραπέμπει σε μοτοσυκλέτα, όπως αντίστοιχα έχει ένα σχολικό λεωφορείο. Δηλαδή τα χαρακτηριστικά του αντικειμένου δεν είναι τόσο μοναδικά από μόνα τους για να τα ξεχωρίζουν από άλλα, συνεπώς χρειαζόμαστε περισσότερα σημεία ενδιαφέροντος και μεγαλύτερους περιγραφείς για περισσότερη ακρίβεια.
- ✚ Αν αυξήσουμε το μέγεθος του λεξικού των οπτικών λέξεων (ερώτημα 4.1) σε μέγεθος 80 τα αποτελέσματα ήταν τα εξής:

KNN:

Total_accuracy: 61.53846153846154 %	Total_accuracy: 71.15384615384616 %
k= 5	k= 33
Total_accuracy: 65.38461538461539 %	Total_accuracy: 69.23076923076923 %
k= 7	k= 35
Total_accuracy: 65.38461538461539 %	Total_accuracy: 63.46153846153846 %
k= 9	k= 37
Total_accuracy: 67.3076923076923 %	Total_accuracy: 71.15384615384616 %
k= 11	k= 39
Total_accuracy: 69.23076923076923 %	Total_accuracy: 71.15384615384616 %
k= 13	k= 41
Total_accuracy: 63.46153846153846 %	Total_accuracy: 69.23076923076923 %
k= 15	k= 43
Total_accuracy: 65.38461538461539 %	Total_accuracy: 69.23076923076923 %
k= 17	k= 45
Total_accuracy: 69.23076923076923 %	Total_accuracy: 71.15384615384616 %
k= 19	k= 47
Total_accuracy: 63.46153846153846 %	Total_accuracy: 71.15384615384616 %
k= 21	k= 49
Total_accuracy: 67.3076923076923 %	Total_accuracy: 69.23076923076923 %
k= 23	k= 51
Total_accuracy: 65.38461538461539 %	Total_accuracy: 69.23076923076923 %
k= 25	k= 53
Total_accuracy: 69.23076923076923 %	Total_accuracy: 67.3076923076923 %
k= 27	k= 55
Total_accuracy: 69.23076923076923 %	Total_accuracy: 67.3076923076923 %
k= 29	k= 57
Total_accuracy: 67.3076923076923 %	Total_accuracy: 67.3076923076923 %
k= 31	k= 59

SVM:

```
Total_accuracy: 80.76923076923077 %
Accuracy_class_motorbikes: 70.0 %
Accuracy_class_school_bus: 66.66666666666666 %
Accuracy_class_touring_bike: 90.9090909090909 %
Accuracy_class_airplane: 100.0 %
Accuracy_class_car_side: 72.72727272727273 %
```

Για μέγεθος : 40

KNN

Total_accuracy: 75.0 %	k= 33
k= 5	Total_accuracy: 69.23076923076923 %
Total_accuracy: 61.53846153846154 %	k= 35
k= 7	Total_accuracy: 71.15384615384616 %
Total_accuracy: 65.38461538461539 %	k= 37
k= 9	Total_accuracy: 67.3076923076923 %
Total_accuracy: 63.46153846153846 %	k= 39
k= 11	Total_accuracy: 71.15384615384616 %
Total_accuracy: 65.38461538461539 %	k= 41
k= 13	Total_accuracy: 71.15384615384616 %
Total_accuracy: 65.38461538461539 %	k= 43
k= 15	Total_accuracy: 71.15384615384616 %
Total_accuracy: 63.46153846153846 %	k= 45
k= 17	Total_accuracy: 69.23076923076923 %
Total_accuracy: 63.46153846153846 %	k= 47
k= 19	Total_accuracy: 69.23076923076923 %
Total_accuracy: 69.23076923076923 %	k= 49
k= 21	Total_accuracy: 69.23076923076923 %
Total_accuracy: 71.15384615384616 %	k= 51
k= 23	Total_accuracy: 67.3076923076923 %
Total_accuracy: 71.15384615384616 %	k= 53
k= 25	Total_accuracy: 65.38461538461539 %
Total_accuracy: 69.23076923076923 %	k= 55
k= 27	Total_accuracy: 67.3076923076923 %
Total_accuracy: 71.15384615384616 %	k= 57
k= 29	Total_accuracy: 65.38461538461539 %
Total_accuracy: 69.23076923076923 %	k= 59
k= 31	

SVM:

```
Total_accuracy: 78.84615384615384 %
Accuracy_class_motorbikes: 50.0 %
Accuracy_class_school_bus: 77.77777777777779 %
Accuracy_class_touring_bike: 90.9090909090909 %
Accuracy_class_airplane: 90.9090909090909 %
Accuracy_class_car_side: 81.81818181818183 %
```

Για μέγεθος 60:

KNN:


```

Total_accuracy: 65.38461538461539 %
k= 5
Total_accuracy: 67.3076923076923 %
k= 7
Total_accuracy: 67.3076923076923 %
k= 9
Total_accuracy: 69.23076923076923 %
k= 11
Total_accuracy: 65.38461538461539 %
k= 13
Total_accuracy: 65.38461538461539 %
k= 15
Total_accuracy: 67.3076923076923 %
k= 17
Total_accuracy: 67.3076923076923 %
k= 19
Total_accuracy: 67.3076923076923 %
k= 21
Total_accuracy: 69.23076923076923 %
k= 23
Total_accuracy: 69.23076923076923 %
k= 25
Total_accuracy: 67.3076923076923 %
k= 27
Total_accuracy: 65.38461538461539 %
k= 29
Total_accuracy: 67.3076923076923 %
k= 31

```

```

Total_accuracy: 69.23076923076923 %
k= 33
Total_accuracy: 69.23076923076923 %
k= 35
Total_accuracy: 65.38461538461539 %
k= 37
Total_accuracy: 65.38461538461539 %
k= 39
Total_accuracy: 63.46153846153846 %
k= 41
Total_accuracy: 63.46153846153846 %
k= 43
Total_accuracy: 61.53846153846154 %
k= 45
Total_accuracy: 63.46153846153846 %
k= 47
Total_accuracy: 63.46153846153846 %
k= 49
Total_accuracy: 63.46153846153846 %
k= 51
Total_accuracy: 65.38461538461539 %
k= 53
Total_accuracy: 65.38461538461539 %
k= 55
Total_accuracy: 63.46153846153846 %
k= 57
Total_accuracy: 65.38461538461539 %
k= 59

```

SVM:

```

Total_accuracy: 78.84615384615384 %
Accuracy_class_motorbikes: 50.0 %
Accuracy_class_school_bus: 77.77777777777779 %
Accuracy_class_touring_bike: 90.9090909090909 %
Accuracy_class_airplane: 100.0 %
Accuracy_class_car_side: 72.72727272727273 %

```

✚ Παρατηρούμε ότι η αύξηση του λεξικού δεν σημαίνει απαραίτητα καλύτερη ακρίβεια. Αυτό μπορεί να εξηγηθεί με ένα φαινόμενο που ονομάζεται *overfitting*. Ο αλγόριθμος γίνεται αρκετά συγκεκριμένος στο dataset των trained εικονών και άρα δεν μπορεί να γίνει γενίκευση. Αυτό σημαίνει ότι η ακρίβεια πέφτει σε εικόνες που δεν ανήκουν στο dataset και άρα πρέπει να βρούμε μία μέση λύση για το μέγεθος του λεξικού με δοκιμές. Στη δική μας περίπτωση για τον knn ήταν το 40, ενώ για τον svm το 50 και το 80, αλλά κρατάμε το 50 για καλύτερη απόδοση. Συνολικά όμως το 40 δίνει τη καλύτερη ακρίβεια.