Όραση Υπολογιστών: Τεχνική Αναφορά 3ης Εργασίας

Παπαδόπουλος Αριστείδης Α.Μ.: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)
```

Εικ.1 Ο ορισμός των συναρτήσεων που χρησιμοποιήθηκαν για την εξαγωγή χαρακτηριστικών και την κωδικοποίηση των εικόνων.

- **Παραγωγή Λεξικού – Κωδικοποίηση:**

Το λεξικό που παράγεται βασίζεται στο μοντέλο Bag Of Visual Words(BOVW),με την χρήση του K-means Clustering.Στο μοντέλο BOVW,κάθε εικόνα περιγράφεται από ορισμένα χαρακτηριστικά, τα οποία προκύπτουν μέσω του K-μεσαίων όρων(K-means Clustering),όπου K ο αριθμός των ομάδων με την κάθε ομάδα να αντιπροσωπεύει ένα χαρακτηριστικό.

```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
```

Εικ.2 Η υλοποίηση του μοντέλου Bag of visual words.

Τα χαρακτηριστικά εντοπίζονται χρησιμοποιώντας περιγραφείς και κρίσιμα σημεία, συγκεκριμένα υλοποιείται SIFT περιγραφέας.Έπειτα εκτελείται ο αλγόριθμος του K-means Clustering,που αποτελείται από τα εξής βήματα:

1. Κάθε διάνυσμα που προκύπτει από τον περιγραφέα SIFT,τοποθετείται στον χώρο, με διαστάσεις όσο και το μέγεθος του διανύσματος. Στην προκειμένη περίπτωση το μέγεθος που προκύπτει από τον SIFT είναι 128.Δηλαδή,κάθε σημείο στον χώρο περιγράφεται από τόσες συντεταγμένες, όσες και το μέγεθος του διανύσματος.
2. Τα διανύσματα που βρίσκονται σε παραπλήσιες θέσεις, ομαδοποιούνται, με κάθε ομάδα να αναφέρεται σε ένα χαρακτηριστικό για τις συγκεκριμένες εικόνες.
3. Η ομαδοποίηση περιγράφεται στα επόμενα βήματα.Αρχικά, επιλέγονται τυχαία σημεία, K στο πλήθος, που λειτουργούν ως κέντρα για κάθε ομάδα-κλάση.
4. Υπολογίζονται οι αποστάσεις κάθε διανύσματος από αυτά τα κέντρα, και γίνεται ομαδοποίηση με βάση τις k μικρότερες αποστάσεις.
5. Υπολογίζονται οι μέσοι όροι των αποστάσεων για κάθε κλάση, και με βάση αυτούς επιλέγονται νέα κέντρα για κάθε ομάδα και η διαδικασία επαναλαμβάνεται.
6. Όταν σταθεροποιηθούν οι μέσοι όροι σταματά η διαδικασία.

```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
```

Εικ.3 Ο ορισμός της συνάρτησης train.

Έτσι, είναι δυνατή η περιγραφή κάθε εικόνας με ένα διάνυσμα πεπερασμένου μήκους και η κατηγοριοποίηση της σε μία ομάδα-κλάση, με βάση το διάνυσμα αυτό.Έπειτα, ακολουθεί η διαδικασία κωδικοποίησης(encoding).Σε αυτήν την φάση, εντοπίζονται τα χαρακτηριστικά κάθε εικόνας,και στην συνέχεια υπολογίζεται πόσες φορές εντοπίζεται κάθε χαρακτηριστικό με βάση το λεξικό.Το αποτέλεσμα που προκύπτει είναι ένας πίνακας όπου οι γραμμές του είναι οι όλες οι εικόνες του training dataset,οι στήλες του είναι τα χαρακτηριστικά που χρησιμοποιούνται για την περιγραφή κάθε εικόνας, δηλαδή το σύνολο των κλάσεων, οπότε είναι ίσο με K.Το περιεχόμενο που προκύπτει από την κωδικοποίηση θα γίνει αντιληπτό με το παρακάτω παράδειγμα:

Εικ.4 Το αποτέλεσμα της κωδικοποίησης για την 1η εικόνα του train dataset.

Σε κάθε κελί βρίσκεται πόσες φορές(πόσο έντονα) συναντάται το κάθε χαρακτηριστικό στην 1η εικόνα που διαβάστηκε από το dataset.Το K το επέλεξα να είναι ίσο με 60,δηλαδή 60 χαρακτηριστικά(60 στήλες),οπότε και το λεξικό έχει μέγεθος 60 θέσεων.

- **Knn (K nearest neighbors)**

```
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
```

Εικ.5 Υλοποίηση της συνάρτησης που εντοπίζει το πλήθος των κλάσεων, καθώς και το directory τους.

Αρχικά πρέπει να βρεθεί το πλήθος των κλάσεων, προκειμένου να είναι γνωστό σε ποια κλάση θα ταξινομηθεί κάθε query εικόνα.Αυτό επιτυγχάνεται με την συνάρτηση της παραπάνω εικόνας, όπου τοποθετείται σε κάθε λίστα, το όνομα της κλάσης(με βάση το path),και το αντίστοιχο path της.Για να καταστεί δυνατή η χρήση του αλγορίθμου Knn,πρέπει και για τις εικόνες του test dataset να βρεθούν τα χαρακτηριστικά τους και να κωδικοποιηθούν σύμφωνα με το BOVW μοντέλο που περιεγράφηκε στην προηγούμενη ενότητα.Έτσι, αφού ακολουθηθεί η διαδικασία της παραγωγής BOVW και της κωδικοποίησης για τις εικόνες του test dataset,πρέπει για κάθε εικόνα του test dataset,να βρεθούν οι K σε αριθμό κοντινότεροι γείτονες, όπου για τον υπολογισμό της απόστασης χρησιμοποιείται η Ευκλείδεια απόσταση.Έπειτα, η κλάση που έχουν οι περισσότεροι γείτονες, ανατίθεται στο query image.Ο αλγόριθμος υλοποιείται στην παρακάτω εικόνα:

```
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
```

Εικ.6 Υλοποίηση της συνάρτησης που εντοπίζει τους πλησιέστερους γείτονες και την κλάση των περισσότερων γειτόνων.

Αφού υπολογιστούν οι αποστάσεις, ταξινομούνται με βάση την θέση(index) των μικρότερων αποστάσεων, ενώ στην συνέχεια επιλέγονται αυτές που χρειάζεται ο ταξινομητής Knn,δηλαδή οι K μικρότερες.Στην συνέχεια,για κάθε θέση που επιλέχθηκε, υλοποιείται μια αναζήτηση, με σκοπό να βρεθεί η κλάση στην οποία βρίσκεται η εικόνα που αντιστοιχεί στην επιλεγμένη θέση και τοποθετούμε μια "ψήφο" στην ανάλογη θέση της λίστας που μετρά τις ψήφους, όπου κάθε θέση της λίστας αυτής αντιστοιχεί σε μία κλάση.Έτσι, εκεί που βρίσκονται οι περισσότεροι ψήφοι, σε εκείνη την κλάση θα ανήκει το query image.Άρα, βρίσκουμε την θέση των περισσότερων ψήφων και ορίζουμε στο query image την κλάση που αντιστοιχεί σε αυτήν την θέση.

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

```
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
```

Εικ.7 Ορισμός της συνάρτησης που υλοποιεί το multi-class classification training με χρήση SVM.

Η εικονιζόμενη συνάρτηση υλοποιεί την ταξινόμηση με SVM,όπου πρώτα γίνεται έλεγχος αν υπάρχει το directory που θα αποθηκευτούν τα SVM.Η ταξινόμηση γίνεται σύμφωνα με βάση την εξής διαδικασία:

1. Ο χώρος στον οποίο βρίσκονται οι εικόνες(με την μορφή των περιγραφέων) χωρίζεται σε 2 ημιεπίπεδα, στο ένα περιέχονται οι εικόνες της ίδιας κλάσης, ενώ στο άλλο όλες οι υπόλοιπες.Η ευθεία που χωρίζει τα ημιεπίπεδα, επιλέγεται με τρόπο που να μεγιστοποιεί την απόσταση μεταξύ των δειγμάτων και της ευθείας.
2. Σε μορφή κώδικα, το βήμα 1 υλοποιείται ως εξής: Ο πίνακας labels έχει τόσες θέσεις όσες και οι εικόνες του train dataset,όπου τοποθετείται "1" στην θέση της εικόνας που βρίσκεται στην αντίστοιχη κλάση, και "0" στις υπόλοιπες εικόνες που δεν ανήκει υπό εξέταση κλάση.Επομένως ο πίνακας labels δημιουργείται τόσες φορές όσες και οι κλάσεις που έχουν βρεθεί και με αντίστοιχο τρόπο δημιουργούνται τόσα SVM όσα και οι κλάσεις.
3. Γίνεται χρήση όλων των SVM που παράχθηκαν, για κάθε εικόνα του test set με σκοπό την εύρεση του SVM που παράγει την μικρότερη απόσταση

μεταξύ της εικόνας που χρειάζεται ταξινόμηση και της ευθείας που χωρίζει τα ημιεπίπεδα, για κάθε SVM.Η θέση(idx) του SVM με την μικρότερη απόσταση, αντιστοιχεί στην κλάση που θα ταξινομηθεί η query image.

```
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)
```

Εικ.8 Υλοποίηση της διαδικασίας ταξινόμησης με χρήση SVM.

- **Τρόπος αξιολόγησης αποτελεσμάτων κάθε μεθόδου ταξινόμησης**

Ο τρόπος αξιολόγησης που ακολουθείται με σκοπό την εύρεση των σωστών προβλέψεων/ταξινομήσεων είναι ίδια και για τις 2 μεθόδους ταξινόμησης και υλοποιείται σύμφωνα με την παρακάτω εικόνα:

```
    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)
```

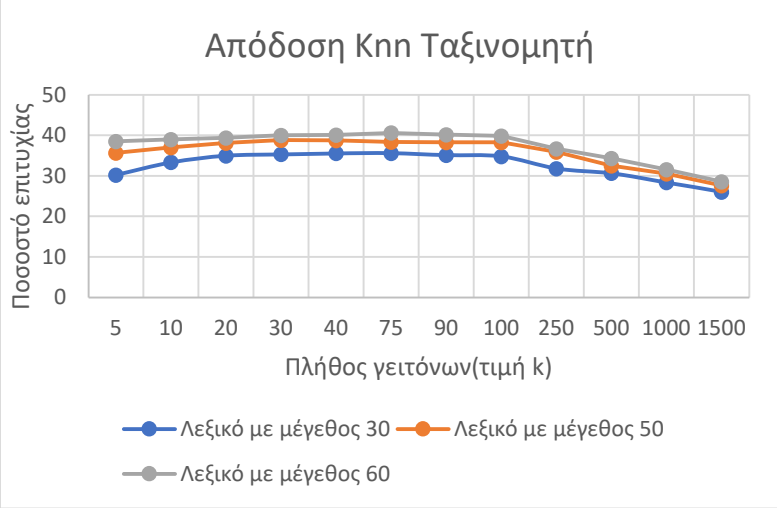
Εικ.9 Υλοποίηση της διαδικασίας αξιολόγησης των ταξινομήσεων.

Για κάθε κλάση που προβλέφθηκε, γίνεται έλεγχος αν αυτή βρίσκεται ταυτόχρονα στην κλάση που εντοπίστηκε με την εκάστοτε μέθοδο ταξινόμησης και αν βρίσκεται στο path της εικόνας που ταξινομείται.Αν ναι,τότε αυξάνεται ένας μετρητής(correct),ενώ για την εύρεση των σωστών ταξινομήσεων ανά κλάση, αυξάνεται η θέση που αντιστοιχεί στην σωστή κλάση, σε έναν πίνακα με τόσες θέσεις όσες και το πλήθος των κλάσεων(class_success).Ακολουθούν πίνακες που παρουσιάζουν τα ποσοστά επιτυχίας για διάφορο πλήθος γειτόνων στην μεθοδολογία ταξινόμησης Knn,αρχικά για μέγεθος λεξικού 30 λέξεων, μετά 50 και τέλος 60:

- Συμπεράσματα για τα αποτελέσματα του Knn ταξινομητή:

Τιμή k(πλήθος γειτόνων)	**Ποσοστό Επιτυχίας(%) για λεξικό 30 θέσεων**	**Ποσοστό Επιτυχίας(%) για λεξικό 50 θέσεων**	**Ποσοστό Επιτυχίας(%) για λεξικό 60 θέσεων**
5	30,24	35,69	38,11
10	33,36	37,04	38,99

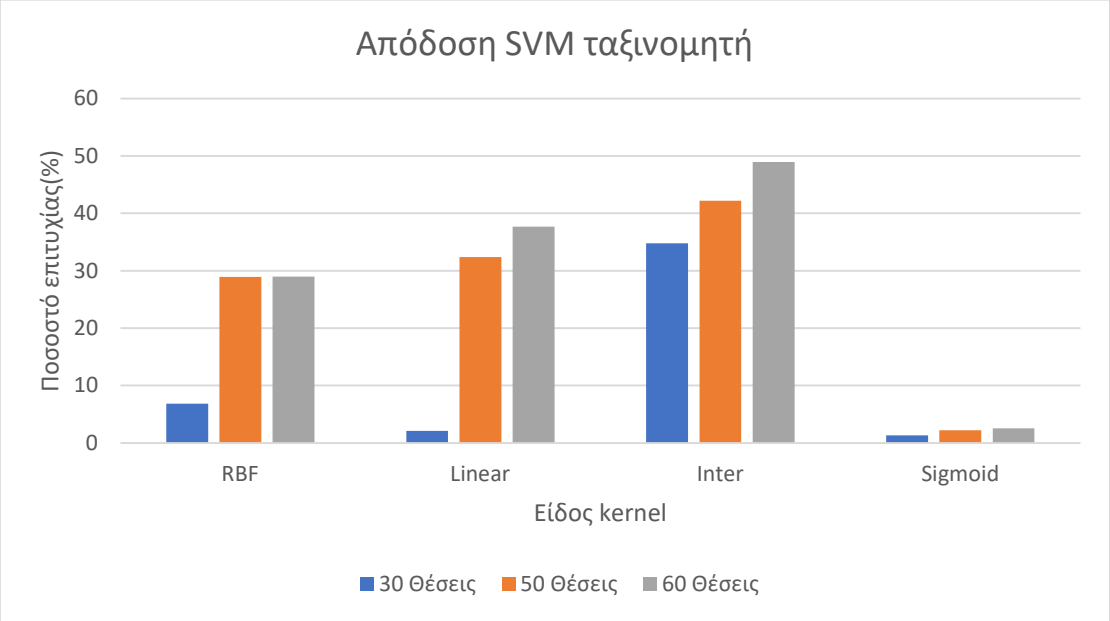
20	34,99	38,11	39,41
30	35,32	38,85	40,02
40	35,55	38,71	40,07
75	35,6	38,39	40,62
90	35,13	38,29	40,2
100	34,80	38,25	39,83
250	31,83	35,87	36,71
500	30,66	32,52	34,3
1000	28,38	30,52	31,6
1500	26,06	27,64	28,57



Λαμβάνοντας υπόψη τα παραπάνω, γίνεται αντιληπτό ότι η αποτελεσματικότητα των ταξινομήσεων εξαρτάται από την τιμή του k σε μεγάλο βαθμό, καθώς επιτυγχάνεται ένα μέγιστο ποσοστό επιτυχίας για μια συγκεκριμένη τιμή k,ενώ στην συνέχεια μειώνεται και φτάνει σε αρκετά χαμηλά ποσοστά.Επίσης, για τιμές k από 30 έως 100,το ποσοστό επιτυχούς πρόβλεψης εμφανίζει μια σταθεροποίηση.Για γείτονες πάνω από 1500 δεν υπολογίστηκε η ευστοχία της υλοποίησης, διότι χάνεται η λειτουργικότητά της.Το παραπάνω φαινόμενο εμφανίζεται, γιατί αρχικά το πλήθος των γειτόνων δεν επαρκούν για την σωστή ταξινόμηση των εικόνων, ενώ για μεγάλο πλήθος γειτόνων, αναλύεται υπερβολικά η διαδικασία και η υλοποίηση πετυχαίνει μόνο για κλάσεις που είχαν πάρα πολλές εικόνες στο αντίστοιχο train dataset.Επίσης γίνεται αντιληπτός ο ρόλος του μεγέθους που έχει το λεξικό, καθώς για την ίδια τιμή γειτόνων k,όσο αυξάνεται το λεξικό, αυξάνεται και το ποσοστό επιτυχίας, αφού το λεξικό έχει μεγαλύτερο μέγεθος.Επομένως, γίνεται αναπαράσταση των εικόνων στο λεξικό με μεγαλύτερο εύρος χαρακτηριστικών, άρα οι εικόνες αναπαρίστανται πιο εύστοχα.Φυσικά, το λεξικό θα μπορούσε να περιέχει και όλα τα χαρακτηριστικά(128 από τον περιγραφέα SIFT),αλλά αυτή η περίπτωση αποτελεί χρονοβόρα και καταναλώνει πόρους, οδηγώντας σε απώλεια της λειτουργικότητας των υλοποιήσεων.

- Συμπεράσματα για την χρήση του SVM ταξινομητή:

Είδος Kernel	Ποσοστό Επιτυχίας(%) για λεξικό 30 θέσεων	Ποσοστό Επιτυχίας(%) για λεξικό 50 θέσεων	Ποσοστό Επιτυχίας(%) για λεξικό 60 θέσεων
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



Σύμφωνα με τα παραπάνω, και σε συνδυασμό με όσα αναφέρθηκαν σχετικά με το μέγεθος του λεξικού, εμφανίζει παρόμοια συμπεριφορά και η ταξινόμηση με SVM.Όσο μεγαλύτερο το μέγεθος του λεξικού, τόσο καλύτερα αποδίδει η υλοποίηση(για το ίδιο είδος kernel).Επίσης, παρατηρείται ότι το καλύτερο είδος πυρήνα είναι το Inter(Histogram Intersection Kernel),με απόδοση που πλησιάζει το 50%,ενώ με διαφορά ο χειρότερος από τους πυρήνες που δοκιμάστηκαν είναι ο Sigmoid,με την απόδοση του να κυμαίνεται κάτω του 3%.Ο Histogram Intersection Kernel(HIK) είναι πιο εύστοχος από τους υπόλοιπους πυρήνες γιατί, ο συγκεκριμένος πυρήνας βασίζεται στην παρουσία χρωμάτων και στα ιστογράμματα που παράγονται από αυτά.Εφόσον οι εικόνες που χρησιμοποιούνται στην εργασία είναι σήματα κυκλοφορίας, έχουν έντονη παρουσία χρωμάτων, επομένως καθιστούν επιτυχή την χρήση του πυρήνα αυτού.Για τον ίδιο λόγο η διαφορά απόδοσης σε σχέση με τους υπόλοιπους kernel είναι αρκετά μεγάλη.Τέλος, για λεξικό μεγέθους 30,όλοι οι πυρήνες αποδίδουν χαμηλό ποσοστό απόδοσης, εκτός από τον HIK,όπου και αυτός όμως αποδίδει αρκετά χαμηλότερα από ότι με λεξικό μεγέθους 50 και 60 λέξεων.

- **Αξιολόγηση αποτελεσμάτων κάθε μεθόδου ταξινόμησης ανά κλάση**

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

Εικ.10 Αξιολόγηση των μεθόδων ταξινόμησης ανά κλάση.

Το ποσοστό ανά κλάση προκύπτει από την διαίρεση των επιτυχών ταξινομήσεων με το πλήθος των εικόνων που δοκιμάστηκαν για κάθε κλάση.Ενδεικτικά ακολουθούν τα αποτελέσματα ανά κλάση, για λεξικό 60 θέσεων, πρώτα με Knn(k=75) και μετά με SVM(με kernel=HIK):

```
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.] %
```

Εικ.11,12 Ποσοστά επιτυχούς ταξινόμησης ανά κλάση των 2 μεθόδων, με Knn(Αριστερά) και SVM(δεξιά).

Από τα αποτελέσματα των παραπάνω εικόνων, φαίνεται πως ο SVM ταξινομητή εμφανίζει μεγαλύτερη ποικιλία στις εύστοχες ταξινομήσεις σε σχέση με τον ταξινομητή Knn,συμπέρασμα αναμενόμενο, εφόσον εμφανίζει μεγαλύτερο ποσοστό επιτυχών ταξινομήσεων.Επίσης, κλάσεις οι οποίες είχαν πολυάριθμες εικόνες στο training data set,έχουν μεγαλύτερη απόδοση από αυτές που είχαν λιγότερες.Ενδεικτικά αναφέρω την κλάση "00032",όπου με τον Knn ταξινομητή το ποσοστό επιτυχούς ταξινόμησης είναι 93,6% ενώ με τον ταξινομητή SVM είναι 87,44%.

Εικ.13 Η πρώτη εικόνα του training dataset για την κλάση “00032”.

Αντίθετα, κλάσεις οι οποίες είχαν λίγες σχετικά εικόνες στο training data set,παρουσιάζουν χαμηλή ή ακόμα και μηδενική απόδοση και με τους 2 ταξινομητές.Για παραδειγματικούς σκοπούς, αναφέρω τις ακόλουθες κλάσεις:

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

Εικ.14,15 Οι πρώτες εικόνες του training dataset για τις κλάσεις “00005” και “00051” αντίστοιχα.

Επιπλέον, κλάσεις με σήματα που είναι μονοχρωματικά, όπως το παράδειγμα που ακολουθεί, παρουσιάζουν εξίσου χαμηλή απόδοση.Συγκεκριμένα, για την κλάση ‘‘00034’’,με την χρήση και των δύο ταξινομητών το αποτέλεσμα είναι μηδενικές σωστές ταξινομήσεις.

Εικ.16 Η πρώτη εικόνα του training dataset για την κλάση “00034”.

Επιπρόσθετα, σήματα που έχουν παραπλήσια αντικείμενα πάνω τους, εμφανίζουν και αυτά χαμηλή απόδοση, με την μεγαλύτερη επιτυχία στις ταξινομήσεις να επιτυγχάνεται για την κλάση με τις περισσότερες εικόνες στο training dataset.Για παράδειγμα, οι κλάσεις ‘‘00007’’,’’00008’’ και ‘‘00010’’ περιέχουν σήματα με την παρουσία ‘‘ανθρώπου’’, και οι περισσότερες επιτυχείς ταξινομήσεις εντοπίζονται στην κλάση ‘‘00007’’,κλάση που έχει και τις περισσότερες εικόνες στο training dataset μεταξύ των τριών:

Κλάση	Ποσοστό με Knn(%)	Ποσοστό με SVM(%)	Αριθμός εικόνων στο train dataset
00007	50	56,67	157
00008	0	0	27
00010	0	21,43	21



Εικ.17,18,19 Οι πρώτες εικόνες στο train dataset για τις κλάσεις “00007”,”00008” και “00010”.

Τέλος, υπάρχουν και μερικές εικόνες στο test dataset που επηρεάζονται από το background και το γύρω περιβάλλον τους, για παράδειγμα έχουν φύλλα από δέντρα μπροστά τους ή αυτοκόλλητα κολλημένα πάνω στις πινακίδες.Επομένως, καθίσταται δύσκολη η ταξινόμηση τους, αφού επηρεάζονται από εξωτερικούς παράγοντες και τελικά υπάρχει μεγάλη πιθανότητα να ταξινομηθούν λάθος τέτοιες εικόνες.



Εικ.20,21,2 Παραδείγματα εικόνων που επηρεάζονται από εξωτερικούς παράγοντες.

- **Τελικά Συμπεράσματα – Σύνοψη**

Η παρούσα υλοποίηση, λόγω των πολλαπλών for-loop και if-statements,χρειάζεται αρκετό χρόνο προκειμένου να τρέξει, ωστόσο μέρος του χρόνου αυτού οφείλεται και στον μεγάλο όγκο εικόνων που χρησιμοποιούνται για το training αλλά και για το testing.Φυσικά, μια διαφορετική υλοποίηση θα είχε ως αποτέλεσμα είτε την αύξηση είτε την μείωση του ποσοστού των επιτυχών ταξινομήσεων.Επίσης, καλύτερα αποτελέσματα θα απέδιδε μια υλοποίηση με spatial pyramids.Ωστόσο, από την μια θεώρησα πως οι spatial pyramids είναι εκτός των ζητουμένων της άσκησης και από την άλλη θεωρώ πως τα αποτελέσματα είναι ικανοποιητικά, ειδικά για την περίπτωση του ταξινομητή SVM,με πυρήνα HIK,όπου η απόδοση είναι σχεδόν 50%.

- **Βιβλιογραφία**
  - 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