ΑΦΜ	ΕΠΩΝΥΜΟ	ΟΝΟΜΑ	ΕΞΑΜΗΝΟ
58352	ΤΟΚΑΤΛΙΔΗΣ	ΓΕΩΡΓΙΟΣ	7^{Ο}

REPORT ΕΡΓΑΣΙΑΣ #3

ΜΑΘΗΜΑ : ΟΡΑΣΗ ΥΠΟΛΟΓΙΣΤΩΝ

ΜΕΡΟΣ Α’

Γενική Επισκόπηση: Σκοπός του 1^{ου} μέρους είναι η ταξινόμηση ζώων σε 5 διακριτές κλάσεις στις οποίες αντιστοιχεί το καθένα με τη χρήση μεθόδων μηχανικής μάθησης. Έχουμε στη διάθεσή μας 1245 δείγματα εκπαίδευσης και 250 δοκιμαστικά δείγματα, ισόποσα μοιρασμένα στις κλάσεις,. Τα βήματα για την αποπεράτωση της ταξινόμησης είναι τα ακόλουθα.

Αρχικά, βρισκόμαστε στη φάση εκπαίδευσης. Παράγουμε το **οπτικό λεξικό** (Bag Of Visual Words) με τη **μέθοδο K-MEANS-CLUSTERING.** Με αυτή τη διαδικασία ορίζουμε πως τα χαρακτηριστικά από τα δείγματά μας θα χωρίζονται σε Κ ομάδες-**clusters**. Τα clusters αυτά είναι ένα σύνολο περιγραφέων (διανυσμάτων με άλλα λόγια) οι οποίοι εξήχθησαν από τις εικόνες εκπαίδευσης και αφορούν τα σημεία-κλειδιά της καθεμιάς. Ανά τακτά διαστήματα **λαμβάνεται και ενημερώνεται το κέντρο κάθε ομάδας** το οποίο είναι η μέση τιμή όλων των διανυσμάτων που ανήκουν στην εκάστοτε ομάδα. Στο τέλος, λαμβάνεται το πλέον πλήρως ενημερωμένο κέντρο τους το οποίο αποτελεί το **πρότυπο οπτικής λέξης**. Με βάση αυτά τα κέντρα σχηματίζεται το **ιστόγραμμα** κάθε δείγματος εκπαίδευσης το οποίο εκφράζεται από

την συχνότητα εμφάνισης των οπτικών λέξεων-κέντρων στην εικόνα, χωρίς να δίνεται έμφαση στη χωρική κατανομή τους. Αυτή η αναπαράσταση είναι αρκετά χρήσιμη για την αποδοτική και πιο «οικονομική» αποθήκευση πληροφορίας, ενώ αξιοποιούνται τελικά και για την εκπαίδευση SVM και knn ταξινομητών μετά από «διανυσματοποίησή» τους και απεικόνισή τους σε έναν πολυδιάστατο διανυσματικό χώρο μαζί με την ετικέτα που διαχωρίζει τις κλάσεις.

Στη φάση των δοκιμαστικών, χρησιμοποιούνται 2 μέθοδοι ταξινόμησης. Η μέθοδος **K-Nearest-Neighbor (knn)** και οι **SVM ταξινομητές**. Η knn μέθοδος βρίσκει τις -K- μικρότερες αποστάσεις του δείγματος δοκιμής από τα δείγματα εκπαίδευσης, δηλαδή τους -K- κοντινότερους γείτονες και το ταξινομεί ανάλογα με το ποια ομάδα μεταξύ των -K- γειτόνων πλειοψηφεί. Ο ταξινομητής SVM αξιοποιεί την αναπαράσταση ιστογραμμάτων με βάση το πώς έχει χωριστεί ο χώρος κατά την εκπαίδευση του SVM, και ταξινομείται το δείγμα ανάλογα με το σε ποια υποπεριοχή θα τοποθετηθεί.

<u>Κώδικας</u>: Ξεκινώντας με τον **κώδικα ομαδοποίησης** – σχηματισμό λεξικού…

```python
import os
import cv2 as cv
import numpy as np

sift = cv.xfeatures2d_SIFT.create()

train_folders = ["mammals/train"]

def extract_local_features(path):
    img = cv.imread(path)
    kp = sift.detect(img)
    desc = sift.compute(img, kp)
    desc = desc[1]
    return desc


# Extract Database
print('Extracting features...')
train_descs = np.zeros((0, 128))
for folder in train_folders:
    subfolders = os.listdir(folder)
    for subfolder in subfolders:
        subfolder = os.path.join(folder, subfolder)
        files = os.listdir(subfolder)
        print(subfolder)
        for file in files:
            path = os.path.join(subfolder, file)
            desc = extract_local_features(path)
            if desc is None:
                continue
            train_descs = np.concatenate((train_descs, desc), axis=0)
```

Φορτώνουμε τις γνωστές βιβλιοθήκες μαζί με την **os** που μας δίνει τη δυνατότητα χρήσης λειτουργιών του λειτουργικού συστήματος. Διατρέχουμε τον φάκελο εκπαίδευσης όπου υπάρχουν οι υποφάκελοι κάθε κλάσης και για κάθε τέτοιο υποφάκελο διατρέχουμε τα στοιχεία-εικόνες που έχουμε βάλει μέσα και εξάγουμε με τον αλγόριθμο SIFT τα χαρακτηριστικά των εικόνων.

Με το πέρας της διαδικασίας αυτής έχουμε αντλήσει και συγκεντρώσει όλους τους περιγραφείς σε μια μεταβλητή (train_descs) και είμαστε έτοιμοι να τους ομαδοποιήσουμε σε ένα λεξικό.

```
33  # Create vocabulary
34  print('Creating vocabulary...')
35  term_crit = (cv.TERM_CRITERIA_EPS, 30, 0.1)
36  trainer = cv.BOWKMeansTrainer(50, term_crit, 1, cv.KMEANS_PP_CENTERS)
37  vocabulary = trainer.cluster(train_descs.astype(np.float32))
38
39  np.save('vocabulary.npy', vocabulary)
```

Με την εντολή της γραμμής 35 ορίζουμε κάποιες παραμέτρους για την εξαγωγή των κέντρων-οπτικών λέξεων, όπως το πόσες φορές θέλουμε να τρέξει ο αλγόριθμος επαναπροσδιορισμού των κέντρων(30 φορές) και με βάση το κριτήριο τερματισμού (που εδώ είναι η ακρίβεια – μεταβολή στη θέση του κέντρου) λέμε πως αν δεν έχουμε μεταβολή της θέσης των κέντρων σε διαδοχικά βήματα πάνω από 0.1, τότε ο αλγόριθμος τερματίζει πριν τις ορισμένες επαναλήψεις.

Τελικά στην γραμμή 36 φτιάχνουμε ένα αντικείμενο-class object(δεν αναφέρεται στις κλάσεις ταξινόμησης) με 50 clusters, κριτήρια που ορίσαμε πριν, τον αριθμό προσπαθειών για εύρεση καλύτερης ομαδοποίησης και τον αλγόριθμο αρχικοποίησης κέντρων (εδώ δεν ακολουθείται η τυχαία μέθοδος, αλλά η μέθοδος k-means++).

Λαμβάνουμε το λεξικό και το σώζουμε σε ξεχωριστό αρχείο.

Συνεχίζουμε με τον **κώδικα σχηματισμού Bag Of Visual Words…**

```
16  def encode_bovw_descriptor(desc, vocabulary):
17      bow_desc = np.zeros((1, vocabulary.shape[0]))
18      for d in range(desc.shape[0]):
19          distances = np.sum((desc[d, :] - vocabulary) ** 2, axis=1)
20          mini = np.argmin(distances)
21          bow_desc[0, mini] += 1
22
23      # Normalization
24      if np.sum(bow_desc) > 0:
25          bow_desc = bow_desc / np.sum(bow_desc)
26      return bow_desc
```

Ο κώδικας δεν διαφοροποιείται ιδιαίτερα σε σχέση με προηγουμένως. Διατρέχουμε κάθε δείγμα εκπαίδευσης. Αυτή τη φορά όμως έχοντας το υπολογισμένο ,εκ των προτέρων, λεξικό σκοπεύουμε να φτιάξουμε -όπως ειπώθηκε και στην γενική επισκόπηση- τα ιστογράμματα κάθε εικόνας, σε μορφή διανύσματος γραμμής 1*K. Αυτή τη διεργασία τελεί

η παραπάνω μέθοδος, όπου για κάθε εικόνα μέσα στο for loop εξάγουμε τους περιγραφείς των χαρακτηριστικών σημείων της εικόνας και **για καθένα από αυτούς** υπολογίζουμε τις αποστάσεις τους από τα κέντρα των clusters. **Αφού βρεθούν όλες οι αποστάσεις για συγκεκριμένο περιγραφέα, ζητάμε τον δείκτη i της μικρότερης απόστασης μέσα στον πίνακα αποστάσεων ο οποίος αντιστοιχεί και στο i-th cluster. Έτσι για τον συγκεκριμένο περιγραφέα γνωρίζουμε πως ανήκει στο i-th cluster. Κατά αυτόν τον τρόπο αυξάνουμε τον μετρητή με δείκτη i ο οποίος προσμετρά τη συχνότητα εμφάνισης της i-th οπτικής λέξης**. Αυτό γίνεται επαναληπτικά για όλους τους περιγραφείς μιας εικόνας λαμβάνοντας τελικά το ιστόγραμμα σε μορφή διανύσματος γραμμής και κατ' επέκταση για όλες τις εικόνες εκπαίδευσης.

```
print('Creating index...')

vocabulary = np.load('vocabulary.npy')

img_paths = []
# train_descs = np.zeros((0, 128))
bow_descs = np.zeros((0, vocabulary.shape[0]))
for folder in train_folders:
    subfolders = os.listdir(folder)
    for subfolder in subfolders:
        subfolder = os.path.join(folder, subfolder)
        files = os.listdir(subfolder)
        print(subfolder)
        for file in files:
            path = os.path.join(subfolder, file)
            desc = extract_local_features(path)
            if desc is None:
                continue
            bow_desc = encode_bovw_descriptor(desc, vocabulary)

            img_paths.append(path)
            bow_descs = np.concatenate((bow_descs, bow_desc), axis=0)

np.save('BOVW.npy', bow_descs)
np.save('paths', img_paths)
```

Στη συνέχεια σώζουμε σε ξεχωριστά αρχεία τα ιστογράμματα αυτά και τις διαδρομές αρχείων μέσα στο σύστημα (όπου μέσα σε αυτά περιέχεται το όνομα της κλάσης - label στην οποία ανήκει το εκάστοτε αντικείμενο).

Τώρα θα περάσουμε στους ταξινομητές μας. Αρχίζοντας με τον **knn**…

Φορτώνουμε το λεξικό μαζί με τα ιστογράμματα και τις διαδρομές συστήματος των εικόνων. Διατρέχουμε τις διαδρομές και βλέπουμε αν μέσα σε αυτές υπάρχουν τα ονόματα των διακριτών κλάσεων. Έτσι γεμίζουμε ένα πίνακα ο οποίος στην ουσία είναι ο πίνακας διαδρομών συστήματος μετατρεπόμενων σε κωδικούς αριθμούς (0,1,2 κ.ο.κ).

Έπειτα κατασκευάζουμε ένα αντικείμενο knn με τα αριθμημένα πλέον labels και τα ιστογράμματα που έχουμε εισάγει και ολοκληρώνουμε την εκπαίδευσή του.

```python
import cv2 as cv
import numpy as np
import os

root = "mammals/test"

bow_descs = np.load('BOVW.npy').astype(np.float32)

img_paths = np.load('paths.npy')

# TRAINING
labels = []
classes = os.listdir(root)

for p in img_paths:
    for i in range(len(classes)):
        if classes[i] in p:
            labels.append(i)

labels = np.array(labels, np.int32)

knn = cv.ml.KNearest_create()
knn.train(bow_descs, cv.ml.ROW_SAMPLE, labels)

```

Ακολουθούμε παρόμοια διαδικασία κατά την φάση δοκιμαστικών. Η γραμμή 41 δημιουργεί ένα αντικείμενο το οποίο θα δεχτεί τους περιγραφείς ενός δοκιμαστικού δείγματος και με την διαδικασία που ακολουθήθηκε και στον προηγούμενο κώδικα θα έχουμε τελικά το ιστόγραμμα της εικόνας υπό δοκιμή (γραμμή 42-46). Έπειτα με το εκπαιδευμένο knn μοντέλο εισάγουμε το ιστόγραμμα σε αυτό και από αυτό λαμβάνουμε διάφορες πληροφορίες με αυτή που μας αφορά να είναι η κλάση στην οποία ταξινομήθηκε. Τελικά συγκρίνεται η απάντηση (response) με την πραγματική κλάση του δείγματος και τυπώνουμε την απάντηση και αν είναι σωστή. Στο τέλος έχουμε και κάποιους μετρητές για να προσδιορίσουμε την απόδοση του μοντέλου.

```python
# TESTING
vocabulary = np.load('vocabulary.npy')
Class = -1
hit_count = np.zeros(len(classes))
hit_ratio = np.zeros(len(classes))
n_of_tests = 0
for objectClass in classes:
    objectFolder = os.path.join(root, objectClass)
    tests = os.listdir(objectFolder)
    Class = Class + 1
    n_of_tests = n_of_tests + len(tests)
    for test in tests:
        test_path = os.path.join(objectFolder, test)

        sift = cv.xfeatures2d_SIFT.create()

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

        img = cv.imread(test_path)
        kp = sift.detect(img)
        bow_desc = descriptor_extractor.compute(img, kp)

        response, results, neighbours ,dist = knn.findNearest(bow_desc, 50)
        response = int (response)
        if classes.index(object
            hit_count[Class] =
            print("It is a " + objectClass + " (Hit)")
        else:
            print("It is a " + classes[response] + " (Miss)")

        cv.imshow('test object',img)
        cv.waitKey()

```

(variable) objectClass: str

```
        hit_ratio[Class] = hit_count[Class] / len(tests)
        print(hit_ratio[Class])

    print(classes)
    print(hit_ratio)
    total_hr = sum(hit_count) / n_of_tests
    print(total_hr)
    pass
```

Ίδια ακριβώς διαδικασία ακολουθείται για την εκπαίδευση και δοκιμή του SVM ταξινομητή.

Μια από τις παραμέτρους που θέτουμε είναι ο τρόπος διαχωρισμού κλάσεων η οποία γίνεται με ένα γραμμικό υπερεπίπεδο, δηλαδή για ένα διανυσματικό χώρο Ν διαστάσεων έχουμε ένα επίπεδο διαχωρισμού διαστάσεων Ν-1(π.χ γραμμή σε ένα δισδιάστατο χώρο ή μια επίπεδη επιφάνεια σε έναν τρισδιάστατο χώρο).

```
sift = cv.xfeatures2d_SIFT.create()

def extract_local_features(path):
    img = cv.imread(path)
    kp = sift.detect(img)
    desc = sift.compute(img, kp)
    desc = desc[1]
    return desc

bow_descs = np.load('BOVW.npy').astype(np.float32)

img_paths = np.load('paths.npy')

# Train SVM
print('Training SVM...')
svm = cv.ml.SVM_create()
svm.setType(cv.ml.SVM_C_SVC)
svm.setKernel(cv.ml.SVM_LINEAR)
svm.setTermCriteria((cv.TERM_CRITERIA_COUNT, 250, 1.e-06))

root = "mammals/test"

classes = os.listdir(root)
labels = []
for p in img_paths:
    for i in range(len(classes)):
        if classes[i] in p:
            labels.append(i)

labels = np.array(labels, np.int32)

svm.trainAuto(bow_descs, cv.ml.ROW_SAMPLE, labels)
svm.save('svm')
```

```python
import cv2 as cv
import numpy as np
import os

sift = cv.xfeatures2d_SIFT.create()

vocabulary = np.load('vocabulary.npy')

root = "mammals/test"
classes = os.listdir(root)

Class = -1
hit_count = np.zeros(len(classes))
hit_ratio = np.zeros(len(classes))
n_of_tests = 0
for objectClass in classes:
    Class = Class + 1
    objectFolder = os.path.join(root, objectClass)
    tests = os.listdir(objectFolder)
    n_of_tests = n_of_tests + len(tests)
    for test in tests:
        test_path = os.path.join(objectFolder, test)
        # Load SVM
        svm = cv.ml.SVM_create()
        svm = svm.load('svm')

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

        img = cv.imread(test_path)
        kp = sift.detect(img)
        bow_desc = descriptor_extractor.compute(img, kp)
```

```python

        response = svm.predict(bow_desc, flags=cv.ml.STAT_MODEL_RAW_OUTPUT)
        i = int (response[1])

        if classes.index(objectClass) == i:
            hit_count[Class] = hit_count[Class] + 1
            print("It is a " + objectClass + " (Hit)")
        else:
            print("It is a " + classes[i] + " (Miss)")

        cv.imshow('test object',img)
        cv.waitKey()

    hit_ratio[Class] = hit_count[Class] / len(tests)
    print(hit_ratio[Class])

print(classes)
print(hit_ratio)
total_hr = sum(hit_count) / n_of_tests
print(total_hr)
pass
```

<u>Αποτελέσματα-Συμπεράσματα</u> :

- **Για 50 clusters**: Για knn(κ=50) και SVM αντίστοιχα

```
['brown_bear', 'camel', 'dolphin', 'giraffe', 'squirrel']
[0.7  0.6  0.56 0.58 0.48]
0.584
```

```
['brown_bear', 'camel', 'dolphin', 'giraffe', 'squirrel']
[0.56 0.56 0.76 0.66 0.44]
0.596
```

- **Για 60 clusters**: Για knn(κ=50) και SVM αντίστοιχα

```
['brown_bear', 'camel', 'dolphin', 'giraffe', 'squirrel']
[0.7  0.48 0.54 0.68 0.4 ]
0.56
```

```
['brown_bear', 'camel', 'dolphin', 'giraffe', 'squirrel']
[0.54 0.5  0.76 0.68 0.48]
0.592
```

- **Για 70 clusters**: Για knn(κ=50) και SVM αντίστοιχα

```
['brown_bear', 'camel', 'dolphin', 'giraffe', 'squirrel']
[0.78 0.58 0.54 0.6  0.46]
0.592
```

```
['brown_bear', 'camel', 'dolphin', 'giraffe', 'squirrel']
[0.6  0.56 0.84 0.68 0.48]
0.632
```

- **Για 80 clusters**: Για knn(κ=50) και SVM αντίστοιχα

```
['brown_bear', 'camel', 'dolphin', 'giraffe', 'squirrel']
[0.76 0.54 0.54 0.64 0.46]
0.588
```

```
['brown_bear', 'camel', 'dolphin', 'giraffe', 'squirrel']
[0.58 0.6  0.76 0.6  0.42]
0.592
```

- **Για 90 clusters**: Για knn(κ=50) και SVM αντίστοιχα

```
['brown_bear', 'camel', 'dolphin', 'giraffe', 'squirrel']
[0.72 0.5  0.56 0.66 0.5 ]
0.588
```

```
['brown_bear', 'camel', 'dolphin', 'giraffe', 'squirrel']
[0.66 0.5  0.76 0.7  0.5 ]
0.624
```

Κρατάμε την επιλογή των 70 cluster και προχωράμε στο επόμενο ερώτημα. Όσο αυξάνεται το κ, παρατηρούμε μια αύξηση στην ακρίβεια. Υπάρχει πάντα μια περιοχή τιμών η οποία παρουσιάζει σταθερή τιμή μέχρι να μεταβούμε σε μεγαλύτερη απόδοση. Η βέλτιστη τιμή παρουσιάζεται για κ=100 με όλες τις άλλες παραμέτρους σταθερές.

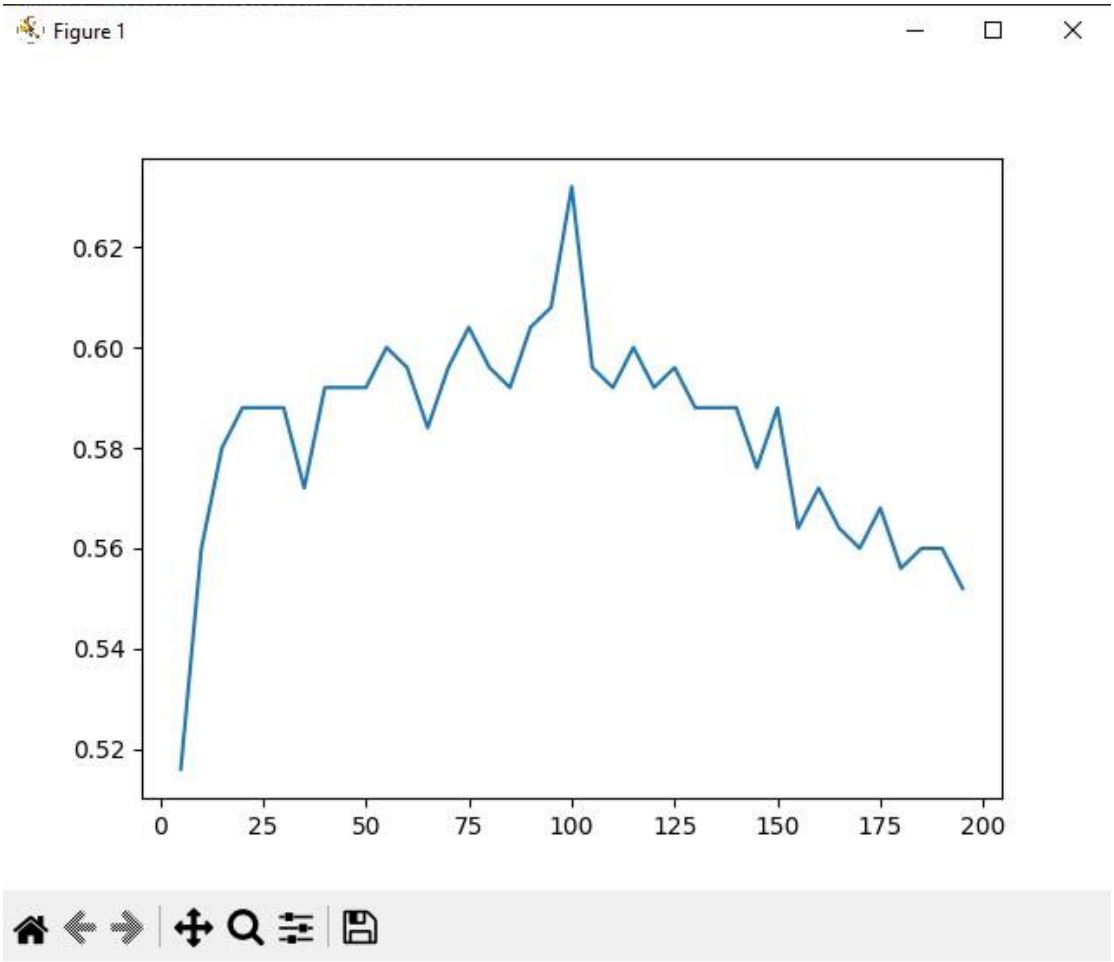
```
['brown_bear', 'camel', 'dolphin', 'giraffe', 'squirrel']
[0.76 0.58 0.52 0.62 0.58]
0.612
```

Αναμένουμε πως για μικρές μέχρι και μεσαίες τιμές Κ όσο αυξάνουμε τον αριθμό Κ δεν έχουμε μια γνησίως αυξητική απόδοση. Αυτό είναι αρκετά λογικό. Ας φανταστούμε το σημείο του δείγματος μας σε ένα δισδιάστατο χώρο και έναν κύκλο με κέντρο το σημείο μας και ακτίνα την απόσταση δείγματος και του κ-οστού γείτονα. Είναι φυσικό για ορισμένα δείγματα τα οποία είναι τοποθετημένα ενδιάμεσα των διαφορετικών ομαδοποιημένων δειγμάτων εκπαίδευσης, να είναι ασαφές σε ποια πραγματική κλάση ανήκει το δείγμα υπό δοκιμασία και η διάκριση να βασίζεται στη τυπική απόκλιση που εμφανίζουν η κάθε κλάση και τα δείγματά τους.

Σημαντική παρατήρηση είναι πως για πολύ μεγάλες τιμές αρχίζουμε να έχουμε φθίνουσα πορεία στην απόδοση με μικρές τοπικές αυξήσεις κατά τη πτώση. Ένας πολύ μεγάλος κύκλος κάνει εξίσου ασαφή τη ταξινόμηση και το μοντέλο αρχίζει να κρίνει πιο «χονδρικά», περίπου όπως συμβαίνει στο underfitting .

Άρα γενικά περιμένουμε μια τριγωνοειδή γραμμή με spikes.

Η κορυφή περιμένουμε να μετατοπίζεται αριστερά για μεγαλύτερο αριθμό cluster και δεξιά για μικρότερο αριθμό cluster, λόγω του ότι με την αύξηση των cluster αυξάνεται η λεπτομέρεια που καταγράφεται από τις εικόνες. Άρα αφού ο αριθμός των cluster και ο αριθμός K σχετίζονται με την απόδοση και τη μεταβάλλουν όσο αυξάνονται σε λογικά πλαίσια, περιμένουμε πως accuracy ~ K1 * Clusters1 = K2 * Clusters2



ΜΕΡΟΣ Β’

<u>Γενική Επισκόπηση</u> : Καλούμαστε να κάνουμε την ίδια εργασία με τη χρήση συνελικτικών δικτύων, μη προ εκπαιδευμένο και προ εκπαιδευμένο. Στο μη προ εκπαιδευμένο η αρχιτεκτονική του δικτύου έχει αλλάξει από την αρχική του εργαστηρίου, καθώς όσος πειραματισμός και να γινόταν με τις υπερπαραμέτρους, για να βρεθούν ισορροπημένα αποτελέσματα, το μοντέλο εμφάνιζε underfitting πράγμα που γινόταν αντιληπτό από τις πολύ χαμηλές αποδόσεις στις φάσεις εκπαίδευσης, επαλήθευσης και δοκιμής ταυτόχρονα. Έτσι υλοποιήθηκε ένα πιο πολύπλοκο-πολυεπίπεδο δίκτυο για να λύσει το πρόβλημα της υπερβολικής απλοποίησης των χαρακτηριστικών των εικόνων. Στη συνέχεια οδηγηθήκαμε στο άλλο άκρο, αυτό του overfitting όπου ένα πολύπλοκο σύστημα - overkill για το μέγεθος του dataset – υπερπροσαρμοζόταν στα δείγματα εκπαίδευσης (σαν να τα απομνημόνευε) κάνοντάς το δηλαδή, πιο επιρρεπές στο θόρυβο και τις αχρείαστες λεπτομέρειες των εικόνων. Αυτό φαινόταν από την εξαιρετική απόδοσή του στη φάση εκπαίδευσης και τη παράλληλη στασιμότητα σε επαλήθευση και δοκιμή.

Τελικά λοιπόν καταλήξαμε σε μια μέση κατάσταση, ως προς τη πολυπλοκότητα. Με λίγα λόγια ο βασικός κορμός της αρχιτεκτονικής, δηλαδή ο αριθμός των επιπέδων, έγινε περισσότερο με οδηγό τον πειραματισμό και τα αποτελέσματα που λαμβάνονταν. Ο αριθμός επιπέδων προσαρμόστηκε λοιπόν με 2 κρυφά επίπεδα ενώ το πλάτος κάθε επιπέδου, δηλαδή ο αριθμός νευρώνων σε κάθε επίπεδο, αυξανόταν για την σύλληψη περισσότερων λεπτομερειών πάνω στα δείγματα. Οι ακριβείς τιμές που επιλέχθηκαν βασίστηκαν σε ήδη υπάρχοντα και μεγαλύτερα μοντέλα, όπως αυτά του vgg .

Στον ίδιο άξονα κινήθηκε και το fine tuning των υπερπαραμέτρων σε συνεργασία με κάποιες βασικές αρχές. Η κυριότερη ήταν πως δεν πρέπει το dataset και τα διακριτά subsets να εισαχθούν πολλές φορές στο δίκτυο, για να αποφευχθεί η υπερπροσαρμογή που παρουσιαζόταν και στο πιο πολύπλοκο δίκτυο, αλλά ούτε και πολύ λίγες. Τα subsets να μην είναι ούτε πολύ μικρά, ούτε πολύ μεγάλα, ώστε να απαλλασσόμαστε κατά κάποιο τρόπο από τον ρυθμό σύγκλισης του αλγορίθμου(π.χ. του gradient descend) ο οποίος επηρεάζει την απόδοση βάση του τι επιλέγουμε. Βέβαια αυτές οι επιλογές αντισταθμίζονται από μεγάλο ή μικρό αριθμό εποχών θεωρητικά. Άρα οι δύο παραπάνω βασικές αρχές μπορούν να συγχωνευθούν σε μία. Ένας σημαντικός ακόμα παράγοντας ήταν η μετατροπή των δεδομένων (data augmentation) για να επαυξήσουμε πλασματικά το dataset μας και να επιτρέψουμε το μοντέλο μας να επιτύχει καλύτερη γενίκευση των χαρακτηριστικών μιας εικόνας. Έχοντας καλύψει αυτά το τελευταίο σημαντικό βήμα ήταν η ρύθμιση του learning rate, ώστε ο αλγόριθμος (διαισθητικά μιλώντας) να έχει αρκετό «χρόνο» να συγκλίνει. Το επιπρόσθετο χαρακτηριστικό που άλλαζε από εκεί και μετά ήταν το μέγεθος της εικόνας το οποίο έδινε ένα παραπάνω σπρώξιμο στις μετρικές του συστήματος αλλά δεν είχε καθοριστικό ρόλο.

Για το εκπαιδευμένο δίκτυο χρησιμοποιήθηκε αυτό του εργαστηρίου, το VGG-16.

Κώδικας :

Σε αυτό το τμήμα φορτώνουμε το dataset με τη βοήθεια του template.ipynb και έπειτα τυπώνουμε μια εικόνα για να εξασφαλίσουμε πως η εργασία μεταφόρτωσης πραγματοποιήθηκε επιτυχώς.

```
import zipfile
!rm /content/download
!rm -r /content/mammals/train
!rm -r /content/mammals/test
!wget https://vc.ee.duth.gr:6960/index.php/s/HUtNngKpQctMI1x/download
local_zip = '/content/download'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('/content')
zip_ref.close()

train_dir = '/content/mammals/train'
test_dir = '/content/mammals/test'
```

```
--2024-02-02 10:20:43--  https://vc.ee.duth.gr:6960/index.php/s/HUtNngKpQctMI1x/download
Resolving vc.ee.duth.gr (vc.ee.duth.gr)... 83.212.140.54
Connecting to vc.ee.duth.gr (vc.ee.duth.gr)|83.212.140.54|:6960... connected.
HTTP request sent, awaiting response... 200 OK
Length: 20285846 (19M) [application/zip]
Saving to: ‘download’

download            100%[===================>]  19.35M  6.47MB/s    in 3.0s

2024-02-02 10:20:47 (6.47 MB/s) - ‘download’ saved [20285846/20285846]
```

```
import os

base_dir = '/content/mammals'

train_dir = os.path.join(base_dir, 'train')
test_dir = os.path.join(base_dir, 'test')
```

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

img_path = "/content/mammals/train/camel/camel-0001.jpg"
img = mpimg.imread(img_path)
plt.imshow(img, cmap='gray')
```

Τώρα είναι στιγμή, να κάνουμε data augmentation και ταυτόχρονα να χωρίσουμε το training set 1245 εικόνων σε μικρότερο training set και validation set. Αυτό επιτυγχάνεται με τη παράμετρο validation_split κατά την δημιουργία του ImageDataGenerator. Έτσι μπορούμε να κάνουμε παραγωγή δεδομένων για το testing phase και το validation phase. Λαμβάνουμε τις εικόνες από καθορισμένο φάκελο, ορίζουμε πως σε κάθε βήμα εποχής θα παίρνει 32 δείγματα το μοντέλο μας για την εκπαίδευσή του, διατηρούμε τον χρωματικό κώδικα rgb αντί για grayscale, λόγω του ότι υπάρχει πληροφορία και στο χρώμα, η εικόνα θα έχει διαστάσεις 150X150 και με τη παράμετρο training ορίζουμε ποιο subset θα χρησιμοποιήσει ο data generator από το πλέον χωρισμένο αρχικό dataset. Το seed είναι για το ανακάτεμα των batches για περαιτέρω αποφυγή από φαινόμενα overfitting. Κατά τα ίδια λειτουργούμε και για το validation set.

```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator

split_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.2
)
# --------------------
# Flow training images in batches of 20 using train_datagen generator
# --------------------
train_generator = split_datagen.flow_from_directory(train_dir,
                                                    batch_size=32,
                                                    class_mode='categorical',
                                                    color_mode='rgb',
                                                    target_size=(150,150),
                                                    subset='training',
                                                    seed=42)
# --------------------
# Flow validation images in batches of 20 using test_datagen generator - FIND OPTIMAL VALUES
# --------------------
validation_generator =  split_datagen.flow_from_directory(train_dir,
                                                          batch_size=32,
                                                          class_mode  = 'categorical',
                                                          color_mode='rgb',
                                                          target_size=(150,150),
                                                          subset='validation',
                                                          seed=42)
```

Τώρα μεταβαίνουμε στην αρχιτεκτονική του νευρωνικού δικτύου. Ξεκινάμε με input layer 32 νευρώνες, kernel 3x3 για την εξαγωγή χαρακτηριστικών και φιλτράρισμα ReLU όπου μηδενίζονται τα αρνητικά βάρη. Στο τέλος κάνουμε max pooling για downsampling με παράθυρο 2X2. Έπειτα έχουμε 2 κρυφά επίπεδα με πλάτη 64 και 128 και στη συνέχεια κάνουμε τον πίνακα διάνυσμα γραμμή για να το περάσουμε στα Dense-Fully Connected Layers. Αυτά είναι απαραίτητα διότι στο τέλος πρέπει να υπολογίσουμε για κάθε νευρώνα την έξοδό του με βάση όλα τα βάρη που έχουν υπολογιστεί προηγουμένως για αυτό και επιλέγονται να τοποθετηθούν στην έξοδο. Για το μοντέλο μας πρσαρμόζουμε το learning rate στο 44% του αρχικού (default value = 0.001) και το βάζουμε να τυπώνει απώλειες και απόδοση στο testing & validation phase.

```python
import tensorflow as tf

# Model architecture
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(5, activation='softmax')
])

model.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 0.00044),
              loss = 'categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

Σώζουμε τα βάρη που εμφάνισαν τη καλύτερη απόδοση – έχουν τις μικρότερες απώλειες σε ξεχωριστό αρχείο. Επικαλούμαστε πως αν δεν έχουμε βελτίωση μετά από 8 εποχές σταματά η εκπαίδευση. Τελικά θέτουμε 20 εποχές για την εκπαίδευση και αφήνουμε τα βήματα για εκπαίδευσης και επιβεβαίωσης στο default(δηλαδή να προσαρμοστούν στο μέγεθος των subset μιας και έχουμε τη δικλείδα ασφαλείας του data augmentation).

```
callbacks = []

save_best_callback = tf.keras.callbacks.ModelCheckpoint(f'best_weights.hdf5', save_best_only=True, verbose=1)
callbacks.append(save_best_callback)

early_stop_callback = tf.keras.callbacks.EarlyStopping(patience=8, restore_best_weights=True, verbose=1)
callbacks.append(early_stop_callback)

history = model.fit(train_generator,
                    validation_data=validation_generator,
                    #steps_per_epoch=,
                    epochs=20,
                    #validation_steps=,
                    #verbose=1,
                    callbacks=callbacks)
```

Στο τέλος τρέχουμε το test με το εκπαιδευμένο μοντέλο μας και τυπώνουμε τις γραφικές παραστάσεις απόδοσης και απωλειών στις φάσεις εκπαίδευσης και επιβεβαίωσης για εποπτικούς λόγους.

```
[ ] test_datagen  = ImageDataGenerator(rescale=1./255)
    test_dir = os.path.join(base_dir, 'test')
    test_generator =  test_datagen.flow_from_directory(test_dir,
                                                       batch_size=50,
                                                       class_mode  = 'categorical',
                                                       color_mode='rgb',
                                                       target_size=(150,150))
    loss, acc = model.evaluate(test_generator)
    print(acc)

    Found 250 images belonging to 5 classes.
    5/5 [==============================] - 3s 575ms/step - loss: 0.9254 - accuracy: 0.6880
    0.6880000233650208
```

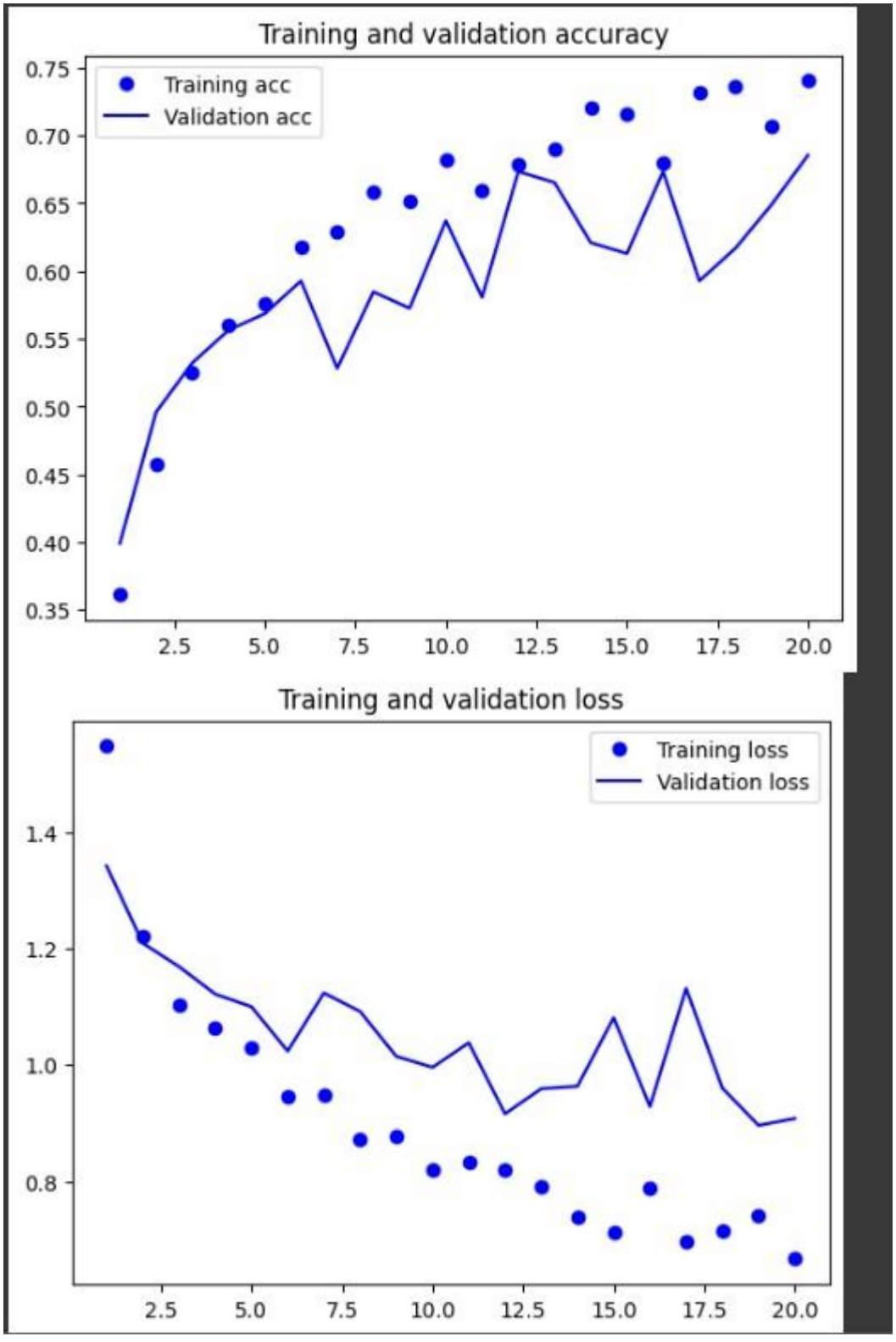
```
    import matplotlib.pyplot as plt

    # Plot training history
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs = range(1, len(acc) + 1)

    plt.plot(epochs, acc, 'bo', label='Training acc')
    plt.plot(epochs, val_acc, 'b', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.legend()

    plt.figure()

    plt.plot(epochs, loss, 'bo', label='Training loss')
    plt.plot(epochs, val_loss, 'b', label='Validation loss')
    plt.title('Training and validation loss')
    plt.legend()

    plt.show()
```

Σχεδόν όμοιος κώδικας και δομή υφίσταται και για το προ εκπαιδευμένο σύστημα Vgg.

Αποτελέσματα:

- **Για το εξατομικευμένο δίκτυο:** 20 epochs of training

```
Found 250 images belonging to 5 classes.
5/5 [==============================] - 3s 575ms/step - loss: 0.9254 - accuracy: 0.6880
0.6880000233650208
```



- **Για το προ εκπαιδευμένο δίκτυο:** 6 epochs for training