Ψηφιακή Επεξεργασία και Ανάλυση Εικόνας
2η Εργασία

Βεργίνης Δημήτριος
1066634

1 Εισαγωγή

Το θέμα που επιλέχθηκε για το 2ο μέρος της εργασίας είναι *Κατηγοριοποίηση Εικόνων – Σύγκριση δύο αντιπροσωπευτικών μεθόδων* και το περιβάλλον υλοποίησης **Python**.

2 Μέρος Α - CNN

2.1 Θεωρητικό Υπόβαθρο

Τα νευρωνικά δίκτυα είναι ένας βασικός τομέας της τεχνητής νοημοσύνης. Όσον αφορά την επεξεργασία εικόνας και γενικότερα οποιουδήποτε πολυδιάστατου σήματος τα βαθιά συνελικτικά δίκτυα(Deep Convolutional Networks) είναι αυτά που χρησιμοποιούνται ευρέως από την επιστημονική κοινότητα.

Ένα απλό Συνελικτικό Νευρωνικό Δίκτυο αποτελείται από διάφορα στάδια τα οποία συνδέονται σε σειρά. Ένα από αυτά τα στάδια αποτελείται από "χάρτες" εισαγωγής δεδομένων(input maps), "χάρτες" χαρακτηριστικών(feature maps) και "χάρτες" άθροισης(pooled maps). Όλοι οι χάρτες είναι διανύσματα 2 διαστάσεων(πίνακες). Ο όρος χάρτης χρησιμοποιείται για να δείξει ότι κάθε επίπεδο έχει πλάτος, ύψος καθώς και βάθος.

Η βασική πράξη σε κάθε επίπεδο του νευρωνικού δικτύου είναι η συνέλιξη με την διαφορά ότι δεν έχουμε μετατόπιση του πυρήνα της συνέλιξης σε βάθος. Εφόσον δεν έχουμε μετατόπιση σε βάθος το αποτέλεσμα της συνέλιξης για

κάθε όγκο δεδομένων είναι η άθροιση της συνέλιξης του κάθε επιπέδου. Έστω τώρα $w_{m,n,k}$ τα βάρη του 2-διάστατου πυρήνα του k-οστού χάρτη όπου m και n είναι οι το ύψος και το πλάτος του πυρήνα. Η συνέλιξη μεταξύ του πυρήνα και μιας συγκεκριμένης τοποθεσίας (x,y) είναι το άθροισμα των βαρών του πυρήνα και τα στοιχεία του χάρτη που συμπίπτουν χωρικά. Πέρα από το άθροισμα των συνελίξεων του κάθε επιπέδου στο τέλος προστίθεται και ένα bias. Τέλος για να μεταφερθούμε στο επόμενο στάδιο περνάμε τα δεδομένα μέσα από μια συνάρτηση ενεργοποίησης(activation function) που συνήθως είναι μη γραμμική συνάρτηση.

Εφόσον τα δεδομένα περάσουν από τη συνάρτηση ενεργοποίησης περνάνε στο επόμενο επίπεδο που είναι οι χάρτες χαρακτηριστικών. Κάθε χάρτης χαρακτηριστικών έχει ένα σύνολο πυρήνα και ένα bias που είναι ξεχωριστά για κάθε χαρακτηριστικό. Στόχος είναι το νευρωνικό δίκτυο να μάθει να να αναγνωρίζει αυτά τα χαρακτηριστικά.

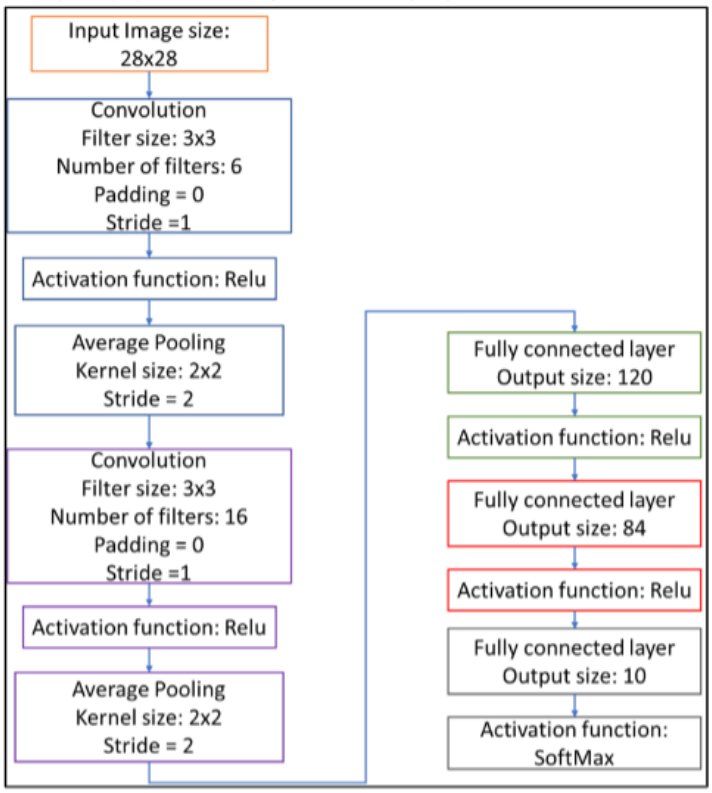
Ύστερα στους χάρτες άθροισης κάθε υποπεριοχή των χαρτών χαρακτηριστικών επεξεργάζεται με κατάλληλο τρόπο(πχ. μέσος όρος) και σχηματίζουν μικρότερου μεγέθους πίνακες.

Στο τέλος οι χάρτες άθροισης μετασχηματίζονται από πίνακες σε μονοδιάστατα διανύσματα και τροφοδοτούνται σε ένα νευρωνικό δίκτυο με στόχο να γίνει η κατηγοριοποίηση ανάλογα πάντα με το πρόβλημα που έχουμε να αντιμετωπίσουμε.

2.2 Στόχος του μέρους Α

Για το συγκεκριμένο πρόβλημα στόχος ήταν η εκπαίδευση ενός συνελικτικού νευρωνικού δικτύου του οποίου η δομή φαίνεται στην παρακάτω εικόνα για την αναγνώριση/κατηγοριοποίηση των χειρόγραφων αριθμών από το 0 έως το 9 από το σετ δεδομένων MNIST. Το MNIST είναι ένα από τα πιο διαδεδομένα σετ δεδομένων που αφορούν την τεχνητή νοημοσύνη και περιλαμβάνει στο σύνολο 70.000 χειρόγραφες εικόνας από ψηφία έχοντας 60.000 από αυτά ως δεδομένα εκπαίδευσης και τα υπόλοιπα 10.000 ως σετ ελέγχου.

Αρχικά τα δεδομένα μας είναι στις διαστάσεις 28x28 και έτσι η είσοδος στα πρώτα συνελικτικά φίλτρα ορίζεται η (28,28,1). Το padding χρησιμεύει στα συνελικτικά δίκτυα έτσι ώστε να διατηρείται το μέγεθος της αρχικής εικόνας καθώς η πράξη της συνέλιξης μειώνει το μέγεθος της εικόνας. Συγκεκριμένα, τα προβλήματα που προκύπτουν με τη συνέλιξη είναι ότι καθώς υπάρχουν αρκετά συνελικτικά φίλτρα, η εικόνα παίρνοντας μέσα από αυτά θα κατέληγε να μην είναι στο αρχικό μέγεθος και δεύτερον τα εικονστοιχεία που βρίσκονται



στην άκρη δεν χρησιμοποιούνται τόσο όσο τα κεντρικά. Για αυτό το λόγο με το padding προσθέτουμε μηδενικά(για το συγκεκριμένο παράδειγμα) στην εικόνα. Το stride είναι κατά πόσα εικονοστοιχεία θα προχωράει σε κάθε βήμα το φίλτρο της συνέλιξης στην εικόνα, εδώ το ορίζουμε να είναι 1 σε κάθε βήμα. Στο τέλος του πρώτου σταδίου εφαρμόζεται στα δεδομένα η συνάρτηση ReLu η οποία ορίζεται ως: $f(x) = max(0, x)$. Μετά περνάνε τα δεδομένα σε ένα επίπεδο Average Pooling όπου τα ανα 4 εικονοστοιχεία υπολογίζεται ο μέσος όρος και σχηματίζεται το νέο επίπεδο. Εφόσον βγουν και από το AveragePooling επίπεδο τροφοδοτούνται ξανά σε ένα σετ συνελικτικών φίλτρων και μετά σε ένα τελευταίο επίπεδο AveragePooling. Σε αυτό το σημείο έχουμε τελειώσει με το κομμάτι του CNN και μετασχηματίζουμε τους πίνακες της εξόδου του τελευταίου επιπέδου σε ένα μονοδιάστατο διάνυσμα για να εισαχθούν σε ένα πλήρες συνδεδεμένο νευρωνικό δίκτυο διαστάσεων (120x84x10) όπου η έξοδος έχει μέγεθος 10, όσα δηλαδή τα ψηφία που θέλουμε να κατηγοριοποιήσουμε. Το διάνυσμα αυτό είναι της μορφής

$[0.00 \quad 0.00 \quad 0.00 \quad 0.00 \quad 0.00 \quad 0.01 \quad 0.01 \quad 0.00 \quad 0.00 \quad 0.98]$.
Έτσι καταλαβαίνουμε σε ποια κλάση ανήκει η κάθε εικόνα βασισμένοι στο μεγαλύτερο στοιχείο του τελικού διανύσματος.

2.3 Υλοποίηση

Όπως αναφέρθηκε στην αρχή η εργασία υλοποιήθηκε σε Python. Για το συγκεκριμένο μέρος χρησιμοποιήθηκαν οι βιβλιοθήκες:

- **Numpy:** Για την φόρτωση των εικόνων ως πίνακες, αριθμητικές πράξεις
- **Pyplot:** Για την σχεδίαση γραφικών παραστάσεων
- **Keras,Tensorflow:** Για την υλοποίηση όλων των μοντέλων νευρωνικών δικτύων
- **Struct:** Για την ανάγνωση των binary αρχείων του MNIST dataset

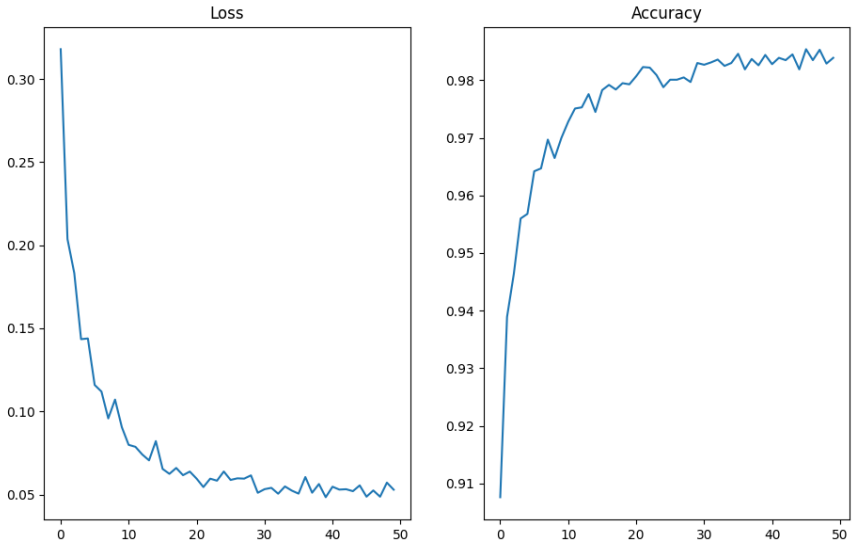
Αρχικά έχοντας κατεβάσει το σετ δεδομένων από το: `http://yann.lecun.com/exdb/mnist/` μέσω της βιβλιοθήκης struct κάνουμε unpack τα δεδομένα του τίτλου και μετά διαβάζοντας σειρά σειρά παίρνουμε τα δεδομένα και τα αποθηκεύουμε σε έναν πίνακα μεγέθους (60000,28,28,1). Την διαδικασία αυτή την κάνουμε για όλα τα δεδομένα(training images, training labels, test images, test labels).Παρακάτω φαίνονται οι απεικονίσεις των ψηφίων από το 0 έως το 9 από τα δεδομένα εκπαίδευσης(κατά τη διάρκεια εκτέλεσης του προγράμματος):

Εφόσον έχουμε φορτώσει τις εικόνες μετατρέπουμε τα labels της κάθε εικόνας σε διανύσματα της μορφής που παρουσιάστηκε παραπάνω έτσι ώστε να μπορεί να τα καταλάβει το νευρωνικό μας δίκτυο. Υλοποιούμε το νευρωνικό δίκτυο που αναφέρθηκε παραπάνω έχοντας ως συνάρτηση σφάλματος την categorical crossentropy που δίνεται από τον τύπο:

$$loss = -\sum_{i=1}^{output-size} y_i log\hat{y}_i$$

όπου το y_i είναι η τιμή στόχος στην συγκεκριμένη περίπτωση διάνυσμα και το $\hat{y}_i$ είναι το διάνυσμα που αναπαριστά την κλάση που μας έδωσε το νευρωνικό δίκτυο ως αποτέλεσμα.

Ύστερα εκπαιδεύουμε το νευρωνικό δίκτυο έχοντας batch_size ίσο με 50 και το εκπαιδεύουμε για 50 εποχές. Τα αποτελέσματα της ακρίβειας και του σφάλματος στα δεδομένα ελέγχου αποθηκεύονται σε αρχεία κάθε φορά που εκτελείται ένα epoch και φαίνονται στις παρακάτω γραφικές παραστάσεις:



Τέλος με τα δεδομένα ελέγχου δημιουργούμε τον τελικό confusion matrix ο οποίος φαίνεται παρακάτω:

Βλέποντας τόσο την ακρίβεια κατά τη διάρκεια των δεδομένων εκπαίδευσης και την ακρίβεια μετά με τα δεδομένα ελέγχου παρατηρούμε ότι η ακρίβεια του μοντέλου που δημιουργήθηκε είναι πάρα πολύ καλή αγγίζοντας το 98.3%. Παρατηρούμε επίσης ότι καθώς και στα δεδομένα ελέγχου έχουμε πολύ καλή ακρίβεια δεν έχουμε το φαινόμενο του overfitting κατά τη διάρκεια της εκπαίδευσης.

```
[[ 972    0    0    0    1    0    4    0    2    1]
 [   0 1126    1    0    0    1    2    0    5    0]
 [   3    0 1014    1    1    0    4    3    6    0]
 [   0    0    1 1001    0    1    0    0    4    3]
 [   0    0    1    0  961    0    5    2    0   13]
 [   2    0    0   17    0  864    3    1    5    0]
 [   0    2    0    1    3    1  950    0    1    0]
 [   1    3    7    1    0    0    0 1010    2    4]
 [   1    0    4    6    1    0    2    2  956    2]
 [   1    2    1    2    5    3    2    5    3  985]]
```

2.4 Κώδικας

```
1
2
3  import numpy as np
4  import struct as st
5  import matplotlib.pyplot as plt
6  import tensorflow
7  import keras
8  from skimage.io import imread
9  from skimage.transform import resize
10 from skimage.feature import hog
11 from skimage import exposure
12 from tensorflow.keras.layers import Dense,Flatten,Conv2D,
       MaxPooling2D,Activation,Input,AveragePooling2D
13 from tensorflow.keras.models import Sequential
14 from PIL import Image
15 import cv2
16 tensorflow.compat.v1.logging.set_verbosity(tensorflow.compat.
       v1.logging.ERROR)
17
18 class TestCallback(keras.callbacks.Callback):
19     def __init__(self, test_data):
20         self.test_data = test_data
21         self.lossFile = open("loss.txt","w")
22         self.accFile = open("acc.txt","w")
23
24     def on_epoch_end(self, epoch, logs={}):
25         x, y = self.test_data
26         loss, acc = self.model.evaluate(x, y, verbose=0)
```

```
27          self.lossFile.write(f'{loss}\n')
28          self.accFile.write(f'{acc}\n')
29
30
31
32
33  def readImages(filename):
34      with open(filename,'rb') as f:
35          magic, size = st.unpack(">II", f.read(8))
36          nrows, ncols = st.unpack(">II", f.read(8))
37          data = np.fromfile(f, dtype=np.dtype(np.uint8).
        newbyteorder('>'))
38          data = data.reshape((size, nrows, ncols))
39      return data
40
41  def readLabels(filename):
42      with open(filename,'rb') as f:
43          magic, size = st.unpack(">II", f.read(8))
44          data = np.fromfile(f, dtype=np.dtype(np.uint8).
        newbyteorder('>'))
45          data = data.reshape((size,))
46      return data
47
48  def showNumbers(labels,images):
49      for i in range(10):
50          ind = np.where(labels == i)[0][0]
51          plt.subplot(1,10,i+1)
52          plt.imshow(images[ind,:,:],cmap='gray')
53          ax = plt.gca()
54          ax.get_xaxis().set_visible(False)
55          ax.get_yaxis().set_visible(False)
56      plt.show()
57
58
59  trainImages = "./MnistDataset/train-images.idx3-ubyte"
60  trainLabels = "./MnistDataset/train-labels.idx1-ubyte"
61  testImages = "./MnistDataset/t10k-images.idx3-ubyte"
62  testLabels = "./MnistDataset/t10k-labels.idx1-ubyte"
63
64  classesNo = 10
65
66  images_train = readImages(trainImages) / 255.0
67  labels_train = readLabels(trainLabels)
68  images_test = readImages(testImages) / 255.0
69  labels_test = readLabels(testLabels)
```

```
70
71 y_train = keras.utils.to_categorical(labels_train, classesNo)
72 y_test = keras.utils.to_categorical(labels_test, classesNo)
73
74 images_train = images_train.reshape((images_train.shape
       [0],28,28,1))
75 images_test = images_test.reshape((images_test.shape
       [0],28,28,1))
76
77
78
79 model = Sequential()
80 model.add(Conv2D(6,kernel_size=(3,3),padding="same",strides
       =1,input_shape=(28,28,1)))
81 model.add(Activation("relu"))
82 model.add(AveragePooling2D(pool_size=(2,2),strides=2))
83 model.add(Conv2D(16,kernel_size=(3,3),padding="same",strides
       =1))
84 model.add(Activation("relu"))
85 model.add(AveragePooling2D(pool_size=(2,2),strides=2))
86 model.add(Flatten())
87 model.add(Dense(120))
88 model.add(Activation("relu"))
89 model.add(Dense(84))
90 model.add(Activation("relu"))
91 model.add(Dense(10))
92 model.add(Activation("softmax"))
93 model.compile(loss=keras.losses.categorical_crossentropy,
94               optimizer=keras.optimizers.SGD(),
95               metrics=['accuracy'])
96
97 model.fit(images_train, y_train,epochs=50, verbose=1,
       batch_size=50,callbacks=[TestCallback((images_test, y_test
       ))])
98
99
100 showNumbers(labels_train,images_train)
101
102
103 loss = open("loss.txt","r")
104 acc = open("acc.txt","r")
105 loss = [float(x) for x in loss.read().splitlines()]
106 acc = [float(x) for x in acc.read().splitlines()]
107
108 plt.subplot(1,2,1)
```

```
plt.plot(loss,"-")
plt.title("Loss")
plt.subplot(1,2,2)
plt.plot(acc,"-")
plt.title("Accuracy")
plt.show()


predictions = model.predict(images_test).argmax(axis=1)

def makeConfusionMatrix(labels,predictions):
    confusionMatrix = np.zeros((10,10),dtype=np.int32)

    for i,j in zip(labels,predictions):
        confusionMatrix[i][j]+=1

    return confusionMatrix

confMatrix = makeConfusionMatrix(labels_test,predictions)
print(confMatrix)
```

3 Μέρος Β - Histograms of Oriented Gradients

3.1 Θεωρητικό Υπόβαθρο

3.1.1 Hogs

Η συγκεκριμένη μέθοδος χρησιμοποιείται για την εξαγωγή χαρακτηριστικών σε μια εικόνα με βάση την ιστογραμμάτων κλίσης. Η βασική ιδέα είναι ότι το σχήμα και η εμφάνιση ενός αντικειμένου μπορεί να χαρακτηριστεί καλά από την κατανομή των τοπικών κλίσεων και των κατευθύνσεων των ακμών χωρίς να έχουμε ακριβή γνώση για αυτές. Στην πράξη αυτό γίνεται χωρίζοντας την εικόνα σε μικρές χωρικές περιοχές, τα λεγόμενα "κελιά όπου για κάθε κελί υπολογίζουμε το τοπικό μονοδιάστατο ιστόγραμμα των κατευθύνσεων των κλίσεων ή του προσανατολισμού των ακμών. Ο συνδυασμός των ιστογραμμάτων αυτών σχηματίζουν την τελική αναπαράσταση.

Αρχικά η κανονικοποίηση των χρωμάτων ως πρώτο βήμα δίνει μέτρια επίδραση στα αποτελέσματα. Ύστερα ο υπολογισμός των κλίσεων γίνεται πρώτα εφαρμόζοντας Γκαουσιανό φίλτρο για την εξομάλυνση της εικόνας και μετά η εφαρμογή μασκών(π.χ Sobel). Το επόμενο βήμα είναι η δημιουργία των

bins για τον κάθε descriptor. Κάθε εικονοστοιχείο υπολογίζει μια σταθμισμένη ψήφο για ένα κανάλι ιστογράμματος προσανατολισμού ακμών με βάση τον προσανατολισμό του στοιχείου κλίσης που βρίσκεται στο κέντρο του και οι ψήφοι συσσωρεύονται σε bins.Τα bins μπορεί να είναι ομοιόμορφα κατανεμημένα από $0° - 180°$ είτε από $0° - 360°$. Η φωτεινότητα και η ισχύς των κλίσεων μπορεί να αλλάξει ανάλογα με το πως επιλέγουμε τη συγχώνευση των κελιών σε μεγαλύτερα μπλοκ και ύστερα κανονικοποιούμε το κάθε μπλοκ ξεχωριστά.

3.1.2 Support Vector Machines

Οι μηχανές διανυσμάτων υποστήριξης είναι μια μέθοδος κατηγοριοποίησης που βασίζεται σε γραμμικούς διαχωρισμούς μέγιστου περιθωρίου, δηλαδή στόχος είναι να βρεθεί το βέλτιστο υπερεπίπεδο που μεγιστοποιεί το κενό η το περιθώριο μεταξύ των κατηγοριών. Επιπλέον μπορεί να χρησιμοποιηθεί το τέχνασμα του πυρήνα για να προσδιοριστεί το βέλτιστο μη γραμμικό όριο αποφάσεων μεταξύ των κατηγοριών.

Έστω ότι έχουμε ένα σύνολο δεδομένων κατηγοριοποιήσης με n σημεία σε έναν d-διάστατο χώρο και επιπλέον υποθέτουμε ότι έχουμε μόνο δυο ετικέτες κατηγοριών. Κάθε υπερεπίπεδο στις d διαστάσεις ορίζεται ως το σύνολο όλων των σημείων $x \in R^d$ που ικανοποιούν την συνάρτηση $h(x) = 0$ όπου $h(x)$ είναι η συνάρτηση υπερεπιπέδου και ορίζεται ως εξής: $h(x) = w^T x + b$. Το διαχωριστικό υπερεπίπεδο διαχωρίζει τον αρχικό χώρο σε δυο ημιχώρους. Έτσι ανάλογα με την τιμή της συνάρτησης του διαχωριστικού υπερεπιπέδου για κάθε σημείου του συνόλου δεδομένων κατηγοριοποίησης αν:

$$h(x) = \begin{cases} 1 & \text{if h(x)} > 0 \\ -1 & \text{if h(x)} < 0 \end{cases}$$

Για να επεκτείνουμε την κατηγοριοποίηση σε m κλάσεις μπορούμε να ακολουθήσουμε δυο προσεγγίσεις κάνοντας χρήση πολλαπλών SVM. Η πρώτη προσέγγιση είναι One-To-One κάνοντας χρήση m SVMs, ένα για κάθε κλάση. Με αυτή την λογική κάθε υπερεπίπεδο χωρίζει τους χώρους μεταξύ δυο κλάσεων αγνοώντας τα σημεία των υπόλοιπων. Στην προσέγγιση One-To-Rest χρησιμοποιούμε $\frac{m(m-1)}{2}$ SVMs και κάθε διαχωριστικό υπερεπίπεδο διαχωρίζει τα σημεία μιας κλάσης με όλες τις υπόλοιπες.

3.2 Υλοποίηση

Όπως αναφέρθηκε στην αρχή η εργασία υλοποιήθηκε σε Python. Για το συγκεκριμένο μέρος χρησιμοποιήθηκαν οι βιβλιοθήκες:

- **Numpy:** Για την φόρτωση των εικόνων ως πίνακες, αριθμητικές πράξεις
- **PIL** Για την αλλαγή του μεγέθους της κάθε εικόνας
- **Struct:** Για την ανάγνωση των binary αρχείων του MNIST dataset
- **Skimage:** Για την υλοποίηση των histograms of oriented gradients
- **Sklearn:** Για την υλοποίηση των SVM

Αρχικά όπως και πριν φορτώσαμε τις εικόνες από τα αρχεία binary αρχεία. Ακολουθήθηκε ακριβως η ίδια διαδικασία με πριν απλά έγινε δυο φορές για να μεταφερθούν τα δυο προγραμμάτα σε διαφορετικά αρχεία. Στη συνέχεια μετατρέψαμε όλες τις εικόνες του σετ δεδομένων σε διαστάσεις (64x128) για να μπορέσουμε να κάνουμε σωστά την υλοποίηση των hogs είτε με patches των 8 ή 16 εικονοστοιχείων. Παίρνοντας τους περιγραφείς με α) 8 pixels per cell και 2x2 cells per block β) 16 pixels per cell με 1 cell per block έγινε αποθήκευση σε binary αρχεία για ευκολία στην διαχείριση του χρόνου του προγράμματος.

Στη συνέχεια έγινε δημιουργία δυο πανομοιότυπων SVM με γραμμικό πυρήνα και την σταθερά κανονικοποίησης $C = 0.1$ και η εκπαίδευση τους πάνω στις ετικέτες των κλάσεων και στους περιγραφείς για τις περιπτώσεις που αναφέρθηκαν παραπάνω. Εφόσον έγινε η εκπαίδευση κάναμε πρόβλεψη με τα δύο αυτά μοντέλα πάνω στους περιγραφείς του σετ ελέγχου και πήραμε για τις δυο περιπτώσεις τα εξής αποτελέσματα: α) ‘98.62% β) 97.87% . Οι confusion matrices για τις δυο αυτές περιπτώσεις φαίνονται παρακάτω:

```
[[ 972    1    1    0    0    0    3    1    2    0]
 [   2 1123    2    2    0    0    2    1    3    0]
 [   0    2 1021    0    0    0    1    7    1    0]
 [   1    0    1  989    1    8    0    4    3    3]
 [   1    4    0    0  962    0    1    1    3   10]
 [   2    0    1   11    0  869    5    0    4    0]
 [   6    3    0    0    3    4  942    0    0    0]
 [   1    4   10    2    5    0    0  991    1   14]
 [   6    1    4    4    0    1    3    2  949    4]
 [   2    7    1    5    8    2    0   11    4  969]]
```

```
[[ 975    1    0    0    0    0    2    1    1    0]
 [   2 1130    0    0    0    0    2    0    1    0]
 [   0    5 1015    0    0    0    1    8    2    1]
 [   0    0    2  999    0    3    0    1    3    2]
 [   0    0    0    0  975    0    3    0    1    3]
 [   2    0    0    5    0  880    4    0    1    0]
 [   4    3    0    0    1    4  944    0    2    0]
 [   0    3    9    1    2    0    0 1006    0    7]
 [   3    1    2    3    1    1    0    1  959    3]
 [   1    4    2    4    7    3    0    8    1  979]]
```

Παρατηρούμε πολύ καλές αποδόσεις και από τα δύο SVMs δίνοντας ένα πολύ μικρό ποσοστό σφαλμάτων στο σετ ελέγχου. Κάνοντας την σύγκριση με το CNN παρατηρούμε ότι το SVM με τα 8 pixels per cell και 2x2 cells per block έχει ελαφρώς καλύτερη απόδοση που μπορούμε να την παραλείψουμε. Όσον αφορά το χρόνο εκπαίδευσης των δυο μοντέλων είχαν παρόμοιο χρόνο για το δικό μου σύστημα ενώ χάνοντας 0.75% ακρίβεια έχουμε πολύ σημαντική διαφορά στην εκπαίδευση και λήψη αποτελεσμάτων.

4 Κώδικας

```
1
2  import numpy as np
3  import struct as st
4  import matplotlib.pyplot as plt
5  import keras
6  from skimage.feature import hog
7
8
9  from PIL import Image
10 from sklearn import metrics,svm
11
12
13 def readImages(filename):
14     with open(filename,'rb') as f:
15         magic, size = st.unpack(">II", f.read(8))
16         nrows, ncols = st.unpack(">II", f.read(8))
17         data = np.fromfile(f, dtype=np.dtype(np.uint8).
       newbyteorder('>'))
18         data = data.reshape((size, nrows, ncols))
19     return data
20
21 def readLabels(filename):
22     with open(filename,'rb') as f:
23         magic, size = st.unpack(">II", f.read(8))
24         data = np.fromfile(f, dtype=np.dtype(np.uint8).
       newbyteorder('>'))
25         data = data.reshape((size,))
26     return data
27
28
29 def resizeData(data):
30     newSize = (64,128)
31     n = data.shape[0]
32     newData = data.reshape((n,28,28))
33     resizedImages = np.zeros((n,newSize[1],newSize[0]))
34     for i in range(n):
35         img = Image.fromarray(newData[i, :, :]).resize(
       newSize)
36         resizedImages[i, :, :] = np.asarray(img)
37
38     return resizedImages
39
40 def getHogDescriptors(data,pixels,cells):
```

```
    n = 128 // pixels
    m = 64 // pixels
    hogDescriptors = np.zeros((data.shape[0],n*m*cells*9))

    for i in range(data.shape[0]):
        hogDescriptors[i,:] = hog(data[i,:], orientations=9,
    pixels_per_cell=(pixels, pixels),
                cells_per_block=(cells, cells),block_norm="
    L1")
    return hogDescriptors


def makeConfusionMatrix(labels,predictions):
    confusionMatrix = np.zeros((10,10),dtype=np.int32)

    for i,j in zip(labels,predictions):
        confusionMatrix[i][j]+=1

    return confusionMatrix


trainImages = "./MnistDataset/train-images.idx3-ubyte"
trainLabels = "./MnistDataset/train-labels.idx1-ubyte"
testImages = "./MnistDataset/t10k-images.idx3-ubyte"
testLabels = "./MnistDataset/t10k-labels.idx1-ubyte"

classesNo = 10

images_train = readImages(trainImages) / 255.0
labels_train = readLabels(trainLabels)
images_test = readImages(testImages) / 255.0
labels_test = readLabels(testLabels)

y_train = keras.utils.to_categorical(labels_train, classesNo)
y_test = keras.utils.to_categorical(labels_test, classesNo)

images_train = images_train.reshape((images_train.shape
    [0],28,28,1))
images_test = images_test.reshape((images_test.shape
    [0],28,28,1))

resizedTrainImages = resizeData(images_train)
resizedTestImages = resizeData(images_test)

# hogTrainDescriptors = getHogDescriptors(resizedTrainImages
```

```
    ,8,2)
# hogTestDescriptors = getHogDescriptors(resizedTestImages
    ,8,2)
# hogTrainDescriptors16_4 = getHogDescriptors(
    resizedTrainImages,16,1)
# hogTestDescriptors16_4 = getHogDescriptors(
    resizedTestImages,16,1)

# file1 = open("hogTrainDescriptors.npy",'wb')
# file2 = open("hogTest,Descriptors.npy",'wb')
# file3 = open("hogTrainDescriptors16_4.npy","wb")
# file4 = open("hogTestDescriptors16_4.npy","wb")
# np.save(file3,hogTrainDescriptors16_4)
# np.save(file4,hogTestDescriptors16_4)
# np.save(file1,hogTrainDescriptors)
# np.save(file2,hogTestDescriptors)

with open("hogTrainDescriptors.npy",'rb') as f:
    hogTrainDescriptors = np.load(f)
with open("hogTestDescriptors.npy",'rb') as f:
    hogTestDescriptors = np.load(f)
with open("hogTrainDescriptors16_4.npy",'rb') as f:
    hogTrainDescriptors16_4 = np.load(f)
with open("hogTestDescriptors16_4.npy",'rb') as f:
    hogTestDescriptors16_4 = np.load(f)

linearKernelSVM16_4 = svm.SVC(kernel='linear', C=0.1)
linearKernelSVM16_4.fit(hogTrainDescriptors16_4,labels_train)

predictionsSVM16_4 = linearKernelSVM16_4.predict(
    hogTestDescriptors16_4)

confMatrixSVM16_4 = makeConfusionMatrix(labels_test,
    predictionsSVM16_4)
print(f'Accuracy:{metrics.accuracy_score(labels_test,
    predictionsSVM16_4)}')
print(confMatrixSVM16_4)

linearKernelSVM8_2 = svm.SVC(kernel='linear', C=0.1)
linearKernelSVM8_2.fit(hogTrainDescriptors,labels_train)

predictionsSVM8_2 = linearKernelSVM8_2.predict(
    hogTestDescriptors)

confMatrixSVM8_2 = makeConfusionMatrix(labels_test,
```

```
    predictionsSVM8_2)
print(f'Accuracy:{metrics.accuracy_score(labels_test,
    predictionsSVM8_2)}')
print(confMatrixSVM8_2)
```