Άσκηση 2

Ψηφιακή επεξεργασία και ανάλυση εικόνας

Περιεχόμενα

Ο κώδικας μπορεί να βρεθεί εδώ:

https://github.com/GrigorisTzortzakis/Digital-image-processing-and-analysis/tree/main

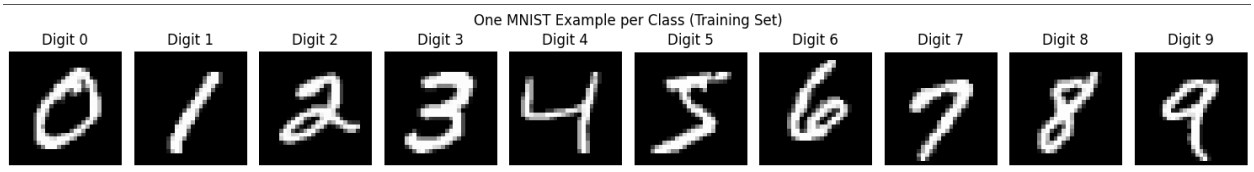
Θέμα 1

Μέρος Α

1)

Αρχικά, το mnist αποτελείτε από 70.000 εικόνες, από τις οποίες 60.000 είναι το training set και 10.000 το test set. Ειδικότερα, αυτές οι εικόνες είναι τα νούμερα 0-9, άρα έχουμε συνολικά 10 διαφορετικές κλάσεις. Συμπερασματικά, έχουμε ένα σύνολο από νούμερα και κάθε εικόνα είναι ένα νούμερο γραμμένο με διαφορετικό τρόπο.

Ο κώδικας υλοποιείτε με python στο περιβάλλον google collab, άρα δεν υπάρχει λόγος να κατεβάσουμε τοπικά το dataset, αφού μπορούμε να έχουμε πρόσβαση σε αυτό με import από έτοιμο library. Παρακάτω βλέπουμε μια εικόνα από κάθε κλάση:



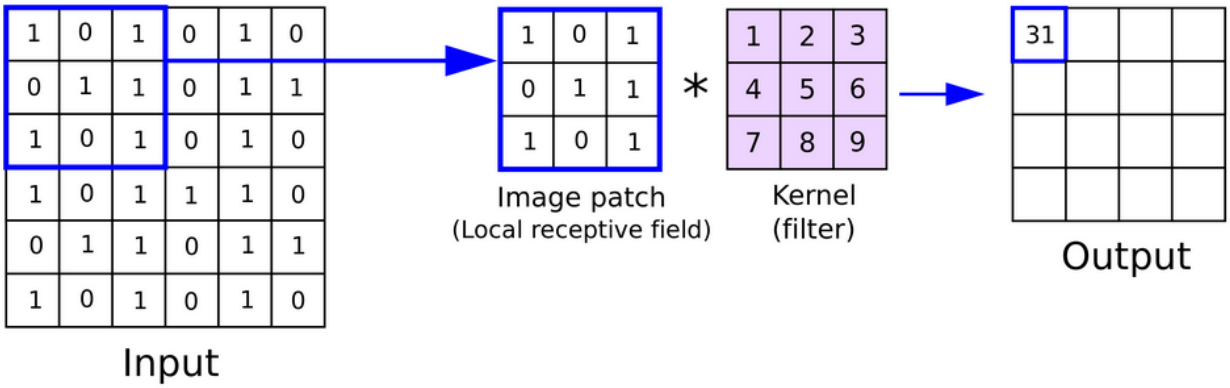
Πριν προχωρήσουμε στο επόμενο ερώτημα, πρέπει πρώτα να κατασκευάσουμε το νευρωσικό δίκτυο. Το πρώτο βήμα είναι το input layer. Αφού οι εικόνες είναι 28χ28, τότε έχουμε διάνυσμα μήκους 784 και άρα θέλουμε 784 input νευρώνες. Στην πραγματικότητα, το input όταν ορίσουμε το batch size αργότερα θα περιγράφεται από έναν tensor.

Προχωρώντας στο πρώτο hidden layer, έχει 6 φίλτρα με συνέλιξη. Αναλυτικότερα, αρχικοποιούμε τα φίλτρα με τυχαία νούμερα στην αρχή, δεν ορίζουμε εμείς από πριν τι θα κάνει το φίλτρο, αφού ο σκοπός είναι να μάθουν από την εικόνα και όχι να τα ορίσουμε εμείς π.χ το ένα φίλτρο είναι:

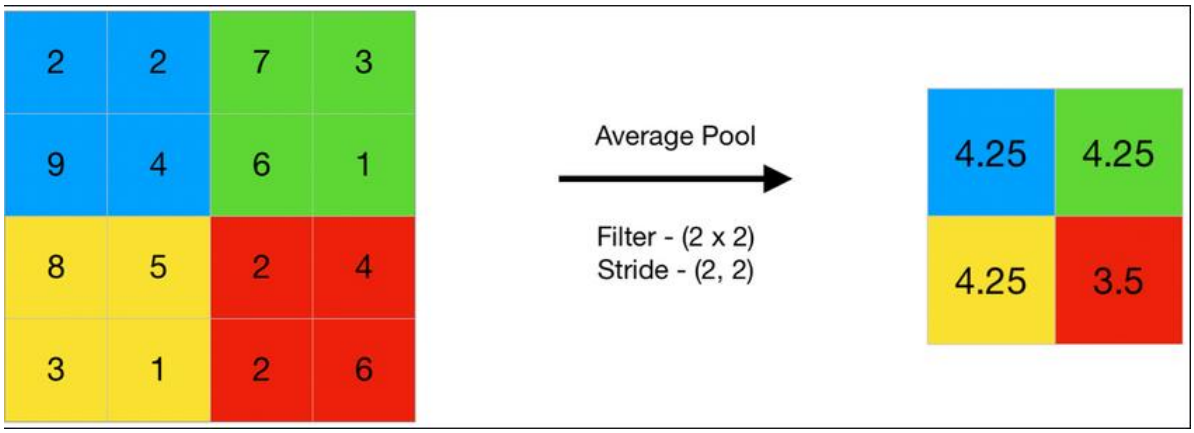
$$\begin{matrix} \mathbf{1} & \mathbf{0} & \mathbf{-1} \\ \mathbf{1} & \mathbf{0} & \mathbf{-1} \\ \mathbf{1} & \mathbf{0} & \mathbf{-1} \end{matrix}$$

Περνάμε το κάθε φίλτρο 3x3 από την εικόνα ώστε να εξάγουμε τις πληροφορίες, δηλαδή ξεκινάμε από το πρώτο pixel της εικόνας και για να υπολογίσουμε την νέα τιμή του πολλαπλασιάζουμε το φίλτρο με την κάθε αντίστοιχη τιμή της εικόνας σε μια 3χ3 περιοχή (δηλαδή 9 pixel σύνολο) και αθροίζουμε το αποτέλεσμα. Αφού έχουμε stride=1, η διαδικασία συνεχίζει

πηγαίνοντας στο ακριβώς δίπλα pixel και συνεχίζει για όλη την γραμμή. Όταν τελειώσει με την γραμμή, συνεχίζει στην επόμενη στήλη το ίδιο μέχρι να έχει διατρέξει όλη την εικόνα. Με συνάρτηση ενεργοποίησης Relu, περνάνε μόνο οι θετικές τιμές στο νέο μητρώο που υπολογίσαμε (όλες οι αρνητικές τιμές μηδενίζονται). Τέλος, αφού έχουμε padding 0 η τελική επιτρεπτή διάσταση χωρίς να φύγουμε εκτός ορίων είναι 26χ26χ6.



Προσθέτοντας, το πρώτο pooling μειώνει τις διαστάσεις μας από 26χ26 σε 13χ13χ6. Αυτό γίνεται επειδή έχουμε 2χ2 kernel, δηλαδή σε αυτή την περιοχή 4 pixel μετατρέπονται σε μια νέα τιμή, η οποία προέρχεται από τον μέσο όρο τους.

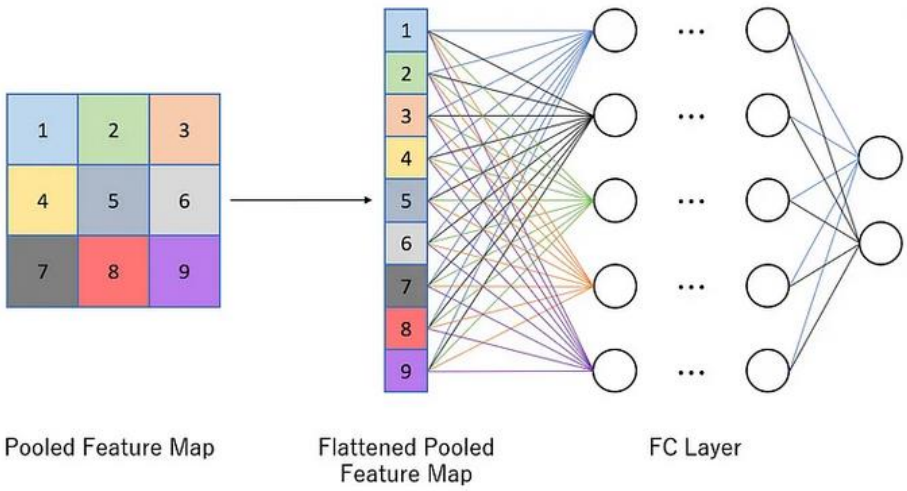


Στο επόμενο συνεκτικό layer γίνεται ακριβώς η ίδια διαδικασία και τώρα οι διαστάσεις μας γίνονται 11χ11χ16. Με το pooling κανονικά οι διαστάσεις μας γίνονται μισές, αλλά στην συγκεκριμένη περίπτωση αυτό είναι 5.5, άρα στρογγυλοποιείτε προς τα κάτω και έχουμε 5χ5χ16.

Κλείνοντας, κάνουμε flatten και κάνουμε map 400 feature σε 120 νευρώνες ώστε να κάνουμε το τελικό scoring. Μεταφέρουμε τα αποτελέσματα σε 84 νευρώνες στο επόμενο layer και στο τελικό σε 10, το οποίο πρακτικά έχει τα τελικά score για κάθε κλάση ώστε να μπορέσουμε να συγκρίνουμε το αποτέλεσμα με το ground truth (για αυτό έχει softmax στο τέλος, μετατρέπει τα αποτελέσματα σε πιθανότητες ώστε να κάνουμε το τελικό prediction).

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 6)	60
average_pooling2d (AveragePooling2D)	(None, 13, 13, 6)	0
conv2d_1 (Conv2D)	(None, 11, 11, 16)	880
average_pooling2d_1 (AveragePooling2D)	(None, 5, 5, 16)	0
flatten (Flatten)	(None, 400)	0
dense (Dense)	(None, 120)	48,120
dense_1 (Dense)	(None, 84)	10,164
dense_2 (Dense)	(None, 10)	850

Total params: 60,074 (234.66 KB)
Trainable params: 60,074 (234.66 KB)



https://medium.com/@alejandro.itoaramendia/convolutional-neural-networks-cnns-a-complete-guide-a803534a1930

https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/

https://www.baeldung.com/cs/neural-networks-conv-fc-layers

https://medium.com/@siddheshb008/lenet-5-architecture-explained-3b559cb2d52b

2)

a)

Αφού εξετάσαμε την αρχιτεκτονική του δικτύου μας, τώρα θα δούμε πως θα εισάγουμε τα δεδομένα σε αυτό ώστε να ξεκινήσουμε την διαδικασία της εκπαίδευσης του. Η διάσπαση των δεδομένων σε batch εξυπηρετεί πολλαπλούς σκοπούς. Πρώτον, αντί να υπολογίσουμε και τις 60.000 εικόνες απευθείας, είναι πιο λογικό να τις χωρίσουμε σε τμήματα και να εκπαιδεύσουμε το σύστημα με αυτό τον τρόπο. Συγκεκριμένα, 1 ολόκληρο batch ισοδυναμεί με μια ολόκληρη εποχή, δηλαδή το σύστημα μας έχει δει όλο το test set μια φορά και τότε ενημερώνει τα βάρη. Όταν το χωρίζουμε σε μικρά τμήματα, μπορούμε να δούμε από τον τύπο 2 ότι ανά εποχή έχουμε πολλαπλές φορές ενημερώσεις βαρών, αφού τα δεδομένα εισέρχονται στο input layer και γίνεται κανονικά όλη η διαδικασία εκπαίδευσης με back propagation κλπ.

Επιπλέον, η διαδικασία της εκπαίδευσης θα συγκλίνει πιο γρήγορα, επειδή οι gpu είναι optimized να δουλεύουν με block. Για παράδειγμα, σε οποιαδήποτε nvidia gpu των τελευταίων 15 ετών, υποστηρίζεται η χρήση cuda, η οποία δέχεται δεδομένα σε block για τον παράλληλο υπολογισμό τους. Επίσης, είναι προφανές ότι απαιτεί και λιγότερους υπολογιστούν πόρους από το να φορτώσουμε όλα τα δεδομένα ταυτόχρονα.

Για το συγκεκριμένο δίκτυο, επιλέγουμε batch size 64 (άρα έχουμε 938 batch σύνολο) και το δημιουργούμε παίρνοντας τυχαία εικόνες από όλες τις κλάσεις. Συνεπώς, το input layer μας έχει διαστάσεις 28x28x64.

https://www.geeksforgeeks.org/mini-batch-gradient-descent-in-deep-learning/

b)

Για να κατανοήσουμε τι είναι το gradient, ας ελέγξουμε τον αλγόριθμο lms. Ο least mean squares είναι μια ειδική περίπτωση του sgd για ένα μόνο δείγμα την φορά. Ο τύπος του είναι :

$$\boldsymbol{w(n+1) = w(n) + \mu e_n x_n}$$

όπου το w(n+1) είναι η επόμενη τιμή που προσπαθούμε να βρούμε, w(n) η τωρινή τιμή που έχουμε, μ το learning step, $\boldsymbol{e_n}$ το σφάλμα μεταξύ πραγματικής τιμής y και της πρόβλεψης y' και $\boldsymbol{x_n}$ η πραγματική τιμή του σήματος. Μπορεί να μην είναι εμφανές στον συγκεκριμένο τύπο, αλλά το gradient υπάρχει σε αυτόν. Ειδικότερα, είναι το $\boldsymbol{e_n x_n}$ και προκύπτει όταν υπολογίσουμε το gradient του πραγματικού σφάλματος, δηλαδή το ½ της διαφοράς τετραγώνων.

Αυτό που αλλάζει για τον stochastic gradient, είναι ο υπολογισμός του gradient. Συγκεκριμένα, αλλάζει η μετρική απώλειας που χρησιμοποιούμε καθώς και ο αριθμός δειγμάτων. Για το συγκεκριμένο ερώτημα, ορίζουμε το cross entropy για την μετρική απώλειας:

$$\boldsymbol{H(p,q) = -\sum_{i=1}^{k} p_i \log(q_i)}$$

όπου το p_i είναι η δυαδική πιθανότητα του πραγματικού label (είναι είτε 0 είτε 1 για να δούμε σε πια κλάση αναφερόμαστε) και το q_i είναι οι πιθανότητες που βρίσκουμε με την softmax. Ο σκοπός μας είναι να ελαχιστοποιήσουμε το συγκεκριμένο σφάλμα.

Συνεπώς, ο stochastic gradient υπολογίζεται από τον τύπο:

$$W(n+1) = W(n) - \mu \frac{1}{N} \sum_{i=1}^{N} (qi - p_i) x_i^T$$

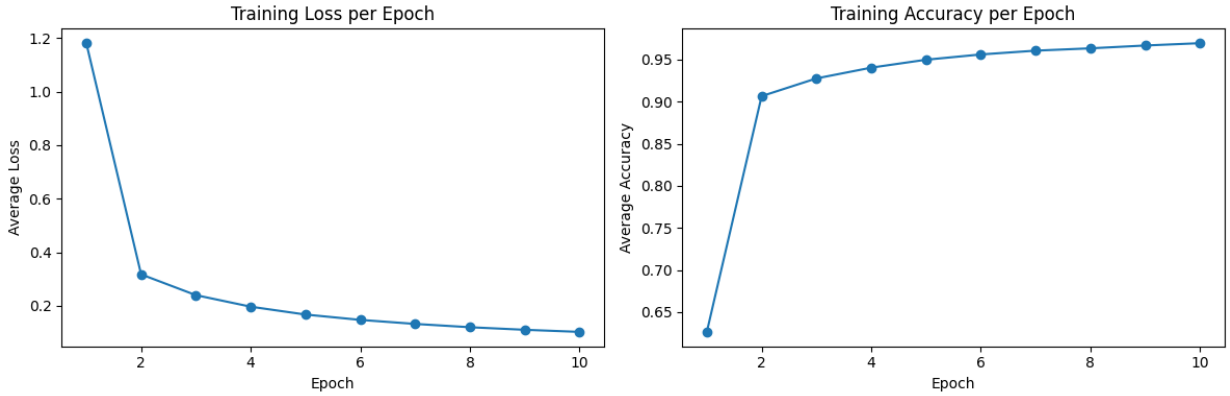
όπου το άθροισμα αυτό προκύπτει αφού υπολογίσουμε το gradient. Εφόσον έχουμε batches, τότε σε αυτό τον τύπο αντικαθιστούμε το N, που είναι ο αριθμός όλων των δειγμάτων με B, που είναι ο αριθμός του batch. Συμπερασματικά, το gradient που υπολογίζουμε είναι πράγματι στοχαστικό, αφού τα batch που διαλέγουμε κάθε φορά είναι τυχαία (είδαμε παραπάνω πως κάθε φορά παίρνουμε 64 τυχαία από όλο το dataset).

```
Step 001 - Mean |gradient|: 0.115770
Step 002 - Mean |gradient|: 0.194315
Step 003 - Mean |gradient|: 0.138750
Step 004 - Mean |gradient|: 0.103553
Step 005 - Mean |gradient|: 0.117791
Step 006 - Mean |gradient|: 0.194389
Step 007 - Mean |gradient|: 0.085521
Step 008 - Mean |gradient|: 0.194587
Step 009 - Mean |gradient|: 0.129832
Step 010 - Mean |gradient|: 0.074402
```

```
Step 928 - Mean |gradient|: 0.103467
Step 929 - Mean |gradient|: 0.165734
Step 930 - Mean |gradient|: 0.211231
Step 931 - Mean |gradient|: 0.171408
Step 932 - Mean |gradient|: 0.091283
Step 933 - Mean |gradient|: 0.108468
Step 934 - Mean |gradient|: 0.119695
Step 935 - Mean |gradient|: 0.149296
Step 936 - Mean |gradient|: 0.096994
Step 937 - Mean |gradient|: 0.126586
Step 938 - Mean |gradient|: 0.133420
```

c)

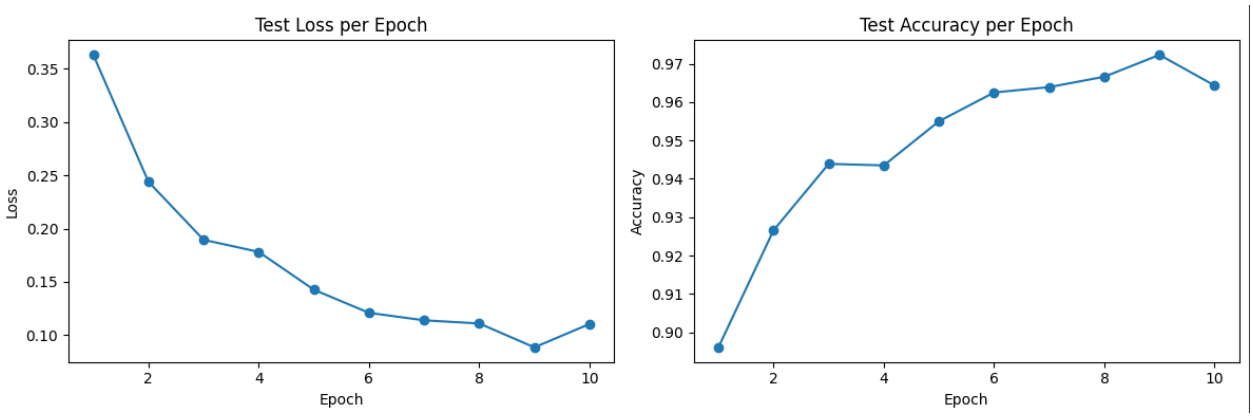
Το μόνο που προσθέτουμε σε σχέση με το προηγούμενο ερώτημα είναι ότι απλά χρησιμοποιούμε ολόκληρο τον τύπο και όχι μόνο το gradient για να βρούμε το σφάλμα και να το μεταδώσουμε πίσω με back propagation. Επιπλέον, θέτουμε 10 εποχές για την διαδικασία της εκπαίδευσης και υπολογίζουμε το loss και το accuracy (Σωστές προβλέψεις/Σύνολο).



Από τα αποτελέσματα βλέπουμε ότι 10 εποχές είναι αρκετές, αφού το σφάλμα πρακτικά έχει συγκλείσει. Βλέπουμε ότι η πρώτη εποχή έχει τεράστιο σφάλμα και χαμηλή ακρίβεια, το οποίο είναι αναμενόμενο αφού πρακτικά το σύστημα μας αρχίζει με τυχαία βάρη και στην συνέχεια διορθώνονται στις επόμενες εποχές. Επίσης, παρατηρούμε ότι όσο μειώνεται το σφάλμα αυξάνεται η ακρίβεια, άρα το σύστημα μας δουλεύει με επιτυχία. Μερικές πτώσεις στην ακρίβεια ανά εποχή οφείλονται στην τυχαιότητα του sgd και είναι πρακτικά αμελητέες

3)

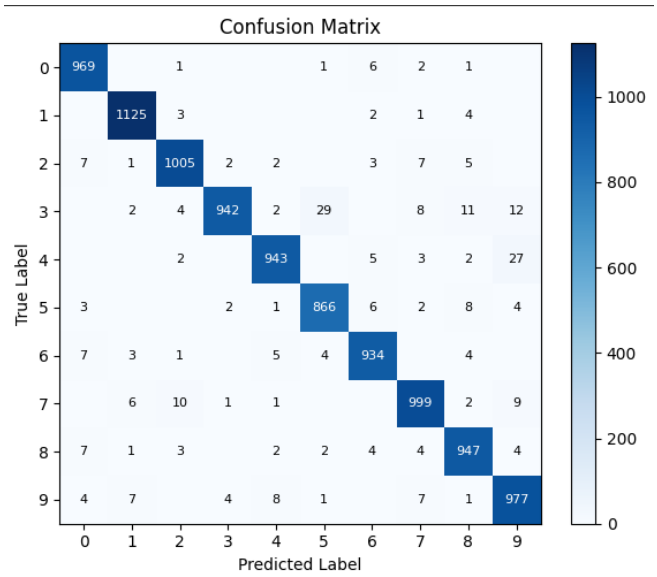
Αφού εκπαιδεύσαμε το σύστημα, τώρα θα εξετάσουμε πόσο καλά γενικεύεται σε δεδομένα τα οποία δεν έχει ξαναδεί. Το μόνο που χρειάζεται να προσθέσουμε στον κώδικα είναι το test dataset και να κάνουμε απλά forward pass, υπολογίζοντας το σφάλμα με κλειδωμένα τα βάρη (δεν επιτρέπεται να αλλάξουν). Είναι πολύ σημαντικό επίσης να κάνουμε reset το μοντέλο μας πριν τρέξουμε τον κώδικα (απλά να ξανατρέξουμε το cell που έχει το model), ώστε να γίνουν reset τα βάρη και να μην αρχίσουμε από το σημείο που τέλειωσε το προηγούμενο ερώτημα (αλλιώς πρακτικά τρέχουμε άλλες 10 εποχές αρχίζοντας από το σημείο που τέλειωσε ο προηγούμενος κώδικας).



Από τα αποτελέσματα δεν βλέπουμε να έχει γίνει overfitting ή underfitting, καθώς το σφάλμα μειώνεται και η ακρίβεια αυξάνεται σταθερά. Οι μικρές αυξομειώσεις σε μερικά σημεία οφείλονται στην τυχαιότητα του αλγορίθμου και τον θόρυβο.

4)

Με το confusion matrix μπορούμε να δούμε ακριβώς τον αριθμό των TP (true positive, σωστή πρόβλεψη), TN (true negative, σωστή πρόβλεψη), FP (false positive, λάθος πρόβλεψη) και FN (false negative, λάθος πρόβλεψη). Άρα, αντί να βλέπουμε μόνο το accuracy εξετάζουμε αναλυτικά τον αριθμό σωστών προβλέψεων του συστήματος μας και σε κάθε label τι λάθη έχουν γίνει.



Το συγκεκριμένο διάγραμμα μας δείχνει αριστερά το ground truth και κάτω την πρόβλεψη που έγινε. Για παράδειγμα, ξεκινώντας από την πρώτη γραμμή (το 0) κοιτάμε κάθε στήλη ώστε να δούμε αναλυτικά τι labeling έγινε. Στην θέση (0,0) βλέπουμε τις σωστές προβλέψεις, στην θέση (0,1) ποσά 0 πρόβλεψε ότι είναι 1, στην θέση (0,2) ποσά 0 πρόβλεψη ότι είναι 2 κπλ. Συμπερασματικά, πετυχαίνουμε πράγματι υψηλή ακρίβεια καθώς το δίκτυο προβλέπει σωστά τις ετικέτες.

Μέρος Β

1)

Η μέθοδος HOG είναι ένας τρόπος να περιγράψουμε τα σχήματα και τις άκρες σε μια εικόνα. Καταρχάς, εξετάζει τμήματα της εικόνας με διάφορα φίλτρα, ώστε να υπολογίσει τα gradients ανά κατεύθυνση. Για παράδειγμα, ένας τρόπος είναι να χρησιμοποιήσουμε 3x3 φίλτρα, όπως είδαμε και στο cnn παραπάνω. Αφού βρούμε το G_x και το G_y, υπολογίζουμε το magnitude (την δύναμη της αλλαγής) και το orientation (την κατεύθυνση της αλλαγής) και υπολογίζουμε ένα 8χ8 ιστόγραμμα. Σε αυτό έχουμε 9 bin, δηλαδή 9 επιλογές για την κατεύθυνση σε μοίρες και τοποθετούμε το κάθε pixel στην κατάλληλη επιλογή. Συγκεκριμένα, η τιμή που εκπροσωπεί κάθε pixel είναι το magnitude και αθροίζουμε όλες τις τιμές σε κάθε bin.

Αφού κάθε κελί 8×8 έχει παράξει το ιστόγραμμά του με τις κατευθύνσεις κλίσης, το επόμενο βήμα είναι να ομαδοποιηθούν αυτά τα κελιά σε επικαλυπτόμενα μπλοκ και να κανονικοποιηθούν τα ιστογράμματά τους. Ένα μπλοκ αποτελείται από ένα πλέγμα 2×2 κελιών, δηλαδή μια περιοχή 16×16. Συγχωνεύοντας τα τέσσερα ιστόγραμμα των 9 bin από αυτά τα κελιά, προκύπτει ένα ενιαίο διάνυσμα 36 διαστάσεων που περιγράφει τις κατευθύνσεις ακμών σε αυτήν την περιοχή των 16×16 pixels. Αυτό το διάνυσμα κανονικοποιείται, έτσι ώστε οι μεταβολές στην τοπική αντίθεση ή στον φωτισμό να μην κυριαρχούν στο σύνολο.

Προχωρώντας, τα μπλοκ επικαλύπτονται, δηλαδή το κάθε μπλοκ μετακινείται κατά ένα κελί (8 pixels) κάθε φορά και έτσι έχουμε πλέον δημιουργήσει τον descriptor μας.

https://medium.com/analytics-vidhya/a-gentle-introduction-into-the-histogram-of-oriented-gradients-fdee9ed8f2aa

https://builtin.com/articles/histogram-of-oriented-gradients

Όλη αυτή η διαδικασία είναι απαραίτητη προκειμένου να χρησιμοποιήσουμε svm μοντέλο. Με το cnn, είδαμε πως αυτό έχει φίλτρα και έτσι μαθαίνει από μόνο του τα χαρακτηριστικά των εικόνων, κάτι το οποίο δεν ισχύει και για τα svm. Η δουλειά του τώρα είναι να δέχεται έτοιμα τα δεδομένα και το μόνο που κάνει είναι να βάζει την κάθε κλάση χωριστά τοποθετώντας την κατάλληλη «γραμμή» ανάμεσα τους, ώστε να έχουν την μέγιστη απόσταση μεταξύ τους.

https://www.geeksforgeeks.org/support-vector-machine-algorithm/

Ο υπολογισμός του Hog και το μοντέλο svm γίνονται με τις εξής συναρτήσεις:

https://www.mathworks.com/help/stats/fitcecoc.html

https://www.mathworks.com/help/vision/ref/extracthogfeatures.html

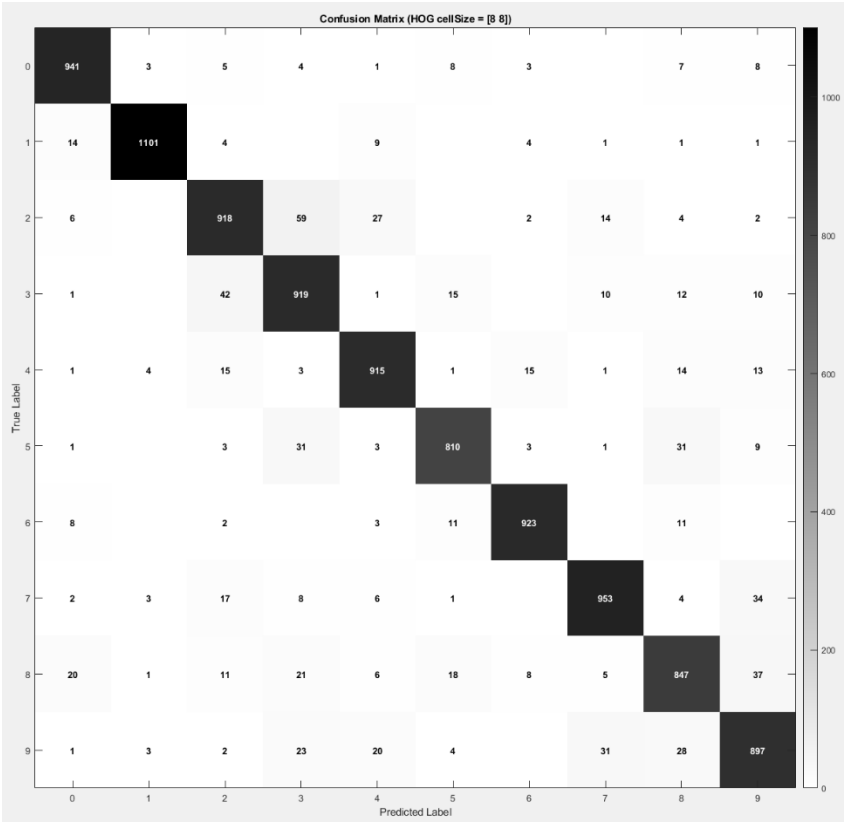
2)

Χρησιμοποιούμε 3 διαφορετικά μεγέθη για το patch. Πρώτα, επιλέγουμε 7χ7 επειδή αυτό καλύπτει ακριβώς τις διαστάσεις της εικόνας (28/7=4 block). Έχουμε επίσης το 8χ8 που είχαμε και στο προηγούμενο ερώτημα, το οποίο δεν καλύπτει ακριβώς τις διαστάσεις της εικόνας και κανονικά χρειάζεται padding. Τέλος, έχουμε 14χ14 το οποίο μας δίνει 2 ακριβώς block για την εικόνα.

```
cellSize = [ 8  8]  →  97.94%
cellSize = [ 7  7]  →  98.95%
cellSize = [14 14]  →  92.24%
```

Από τα αποτελέσματα βλέπουμε ότι το μικρότερο patch είναι το καλύτερο, αφού «εφαρμόζει» ακριβώς στις διαστάσεις της εικόνας και έχει αρκετά block για να καταγράψει αναλυτικά τις αλλαγές ανάμεσα στα pixel. Το κλασσικό μέγεθος 8χ8 δεν έχει μεγάλη διάφορα, αλλά υποφέρει επειδή δεν μπορεί να δει όλα τα pixel της εικόνας. Τέλος, το μεγαλύτερο patch έχει τα χειρότερα αποτελέσματα, καθώς με μόλις 2 block δεν γίνεται να καταγράψει αναλυτικά όλες τις αλλαγές μεταξύ των pixel (περιγράφει το κάθε block 196 pixel, είναι πολύ μεγάλη περιοχή).

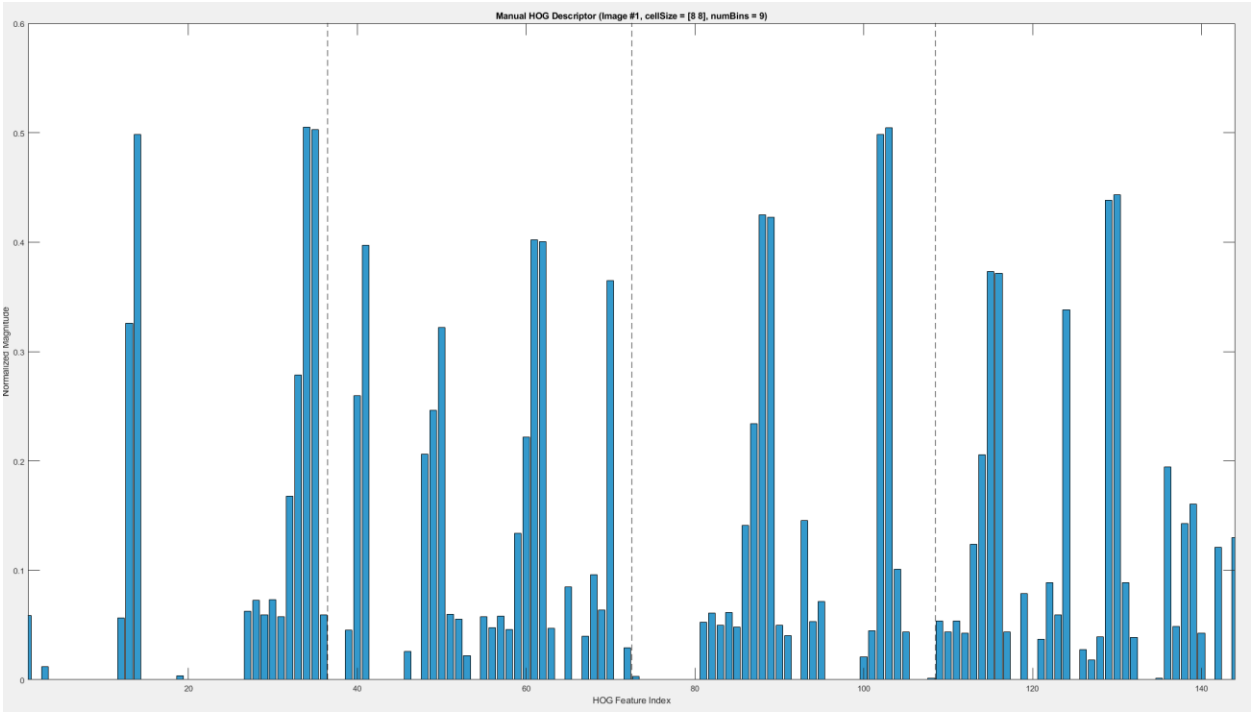
3)



Χρησιμοποιούμε το 8x8 patch και διαπιστώνουμε ότι το svm δεν πετυχαίνει την απόδοση του cnn. Συγκεκριμένα, στο προηγούμενο ερώτημα είδαμε πως το cnn έκανε ελάχιστα λάθη (στο 3 και στο 4 έκανε μόνο σχετικά μεγάλο miss labeling) ενώ τώρα, το svm κάνει λάθος πρόβλεψη σε περισσότερα ψηφία. Σε γενικές γραμμές, η διάφορα τους είναι αρκετά μικρή αλλά και ταυτόχρονα αναμενομένη. Το cnn μαθαίνει τα feature της εικόνας μόνο του, αλλά έχει επιπλέον πολυπλοκότητα και κόστος υλοποίησης. Με το svm, του δίνουμε έτοιμα τα feature και αυτό απλά τα διαχωρίζει σε κατηγορίες, μια διαδικασία που φυσικά είναι πολύ πιο απλή.

4)

Κάνουμε τα βήματα που περιγράψαμε στο ερώτημα 1.



Το dataset για το μέρος β με matlab έγινε download με ένα script, το οποίο δίνεται μαζί με όλο τον υπόλοιπο κώδικα.