Πολυτεχνική Σχολή
Τμήμα Μηχανικών Η/Υ & Πληροφορικής

ΨΗΦΙΑΚΉ ΕΠΕΞΕΡΓΑΣΊΑ ΚΑΙ ΑΝΆΛΥΣΗ ΕΙΚΌΝΑΣ.

ΕΡΓΑΣΤΗΡΙΑΚΉ ΆΣΚΗΣΗ 2

Κατσαρός Ανδρέας
Α.Μ. 1084522

Πάτρα, 2024-25

ΕΡΩΤΗΜΑΤΑ

Σημείωση: *Ο πλήρης κώδικας έχει αποσταλεί σε αρχείο τύπου .ipynb με σχόλια και αναλυτικές εξηγήσεις για κάθε ερώτημα. Κάθε αρχείο περιλαμβάνει αναλυτική παρουσίαση της λογικής και της υλοποίησης του κώδικα. Επιπλέον, η μορφοποίηση στα .ipynb έχει γίνει με τρόπο ώστε κάθε ζητούμενο να παρουσιάζεται καθαρά και οπτικά οργανωμένα.Αντίστοιχα σε .m για το Μέρος Β.*

Μέρος Α:
Κατηγοριοποίηση Εικόνων με χρήση Συνελικτικών Νευρωνικών Δικτύων (Convolutional Neural Networks - CNN)

Ερώτημα 1:

Αρχικά πραγματοποιήθηκε η φόρτωση του συνόλου MNIST στο περιβάλλον του Google Colab μέσω της ενσωματωμένης συνάρτησης keras.datasets.mnist.load_data(), η οποία επέτρεψε την άμεση και αξιόπιστη ανάκτηση των δεδομένων χωρίς επιπλέον ρυθμίσεις δικτύου ή αποθήκευσης. Μ 'αυτό το τρόπο δε χρειάζεται η τοπική εγκατάσταση του MNIST dataset.

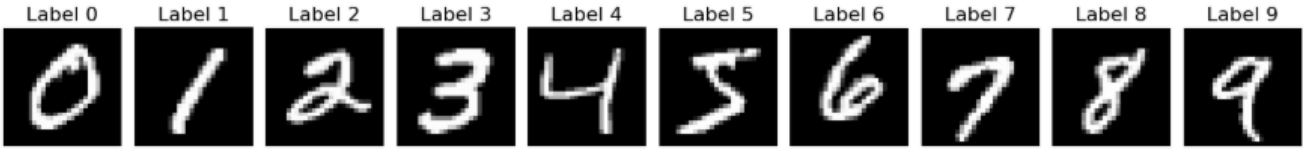
```
# Φορτώνουμε το MNIST από τα έτοιμα σύνολα της Keras
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
```

Έπειτα επιβεβαιώνουμε ότι το σετ εκπαίδευσης περιλαμβάνει 60000 εικόνες διαστάσεων 28×28 pixels και το σετ ελέγχου 10 000 εικόνες ανάλογης διάστασης, και η σωστή φόρτωση των πινάκων pixel και των αντίστοιχων label.

```
Μέγεθος συνόλου εκπαίδευσης: (60000, 28, 28) (60000,)
Μέγεθος συνόλου ελέγχου  : (10000, 28, 28) (10000,)
```

Στη συνέχεια δημιουργήθηκε μια διάταξη 1×10 και μέγεθος 15×3 μονάδων χρησιμοποιώντας την plt.subplots, ώστε να απεικονιστεί από μία αντιπροσωπευτική εικόνα για κάθε ψηφιακή κλάση από το 0 έως το 9.

Για κάθε ψηφίο αντλήθηκε το πρώτο παράδειγμα εμφάνισής του στο σετ εκπαίδευσης (με τη χρήση της np.where), απεικονίστηκε σε grayscale και επισημάνθηκε με τίτλο την αντίστοιχη κλάση.



Έτσι εχουμε μία ενιαία γραμμική οπτικοποίηση των ψηφίων 0–9, επιβεβαιώνοντας οπτικά την επιτυχή ολοκλήρωση της πρώτης φάσης.

Έπειτα εφαρμόστηκε το normalization των τιμών των pixel, μετατρέποντας τους ακέραιους δείκτες από το διάστημα [0, 255] στο εύρος [0, 1] μέσω διαίρεσης με την τιμή 255.0, με στόχο τη διασφάλιση ομαλότερης και ταχύτερης συνάρτησης του SGD κατά το στάδιο της εκπαίδευσης.

```
# 1) float32 & κανονικοποίηση στο [0, 1]
x_train = x_train.astype("float32") / 255.0
x_test  = x_test.astype("float32")  / 255.0
```

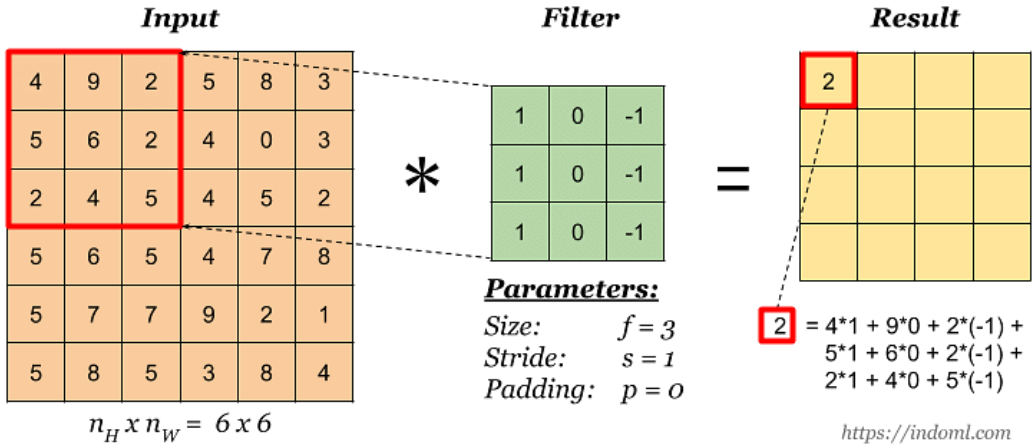
```
X_train: (60000, 28, 28, 1)   y_train_cat: (60000, 10)
```

Πραγματοποιήθηκε η επέκταση της διάστασης των πινάκων εικόνας, προσθέτοντας στο τέλος του σχήματος έναν άξονα depth=1 (normalization & one hot encoding).
Ο έλεγχος της δομής των δεδομένων επιβεβαίωσε ότι το σύνολο εισόδων X_train διαθέτει σχήμα (60000, 28, 28, 1), ενώ το κωδικοποιημένο σετ εξόδων y_train_categ εμφανίζει σχήμα (60000, 10), γεγονός που επαληθεύει τη σωστή προετοιμασία των δεδομένων.

Ερώτημα 2:

Προτού προχωρήσουμε στον ορισμό και την εκπαίδευση του συγκεκριμένου συνελικτικού νευρωνικού δικτύου (CNN), παρουσιάζουμε τη βασική διαδικασία κατασκευής ενός τυπικού CNN.
Σε κάθε CNN βρίσκεται ένα convolutional layer, όπου εφαρμόζεται μία σειρά από φίλτρα (kernels 3×3 ή 5×5 κλπ) πάνω στην είσοδο–εικόνα ή στα αντίστοιχα features από προηγούμενες στρώσεις.
Με την ολίσθηση (stride) του φίλτρου κάθε φορά κατά ένα ή περισσότερα pixel καταγράφεται (dot-product) μεταξύ φίλτρου και τοπικής περιοχής, παράγοντας έτσι έναν χάρτη χαρακτηριστικών (feature map)

Παρακάτω παρουσιάζεται ακριβώς η λογική αυτή:



https://www.projectpro.io/article/introduction-to-convolutional-neural-networks-algorithm-architecture/560

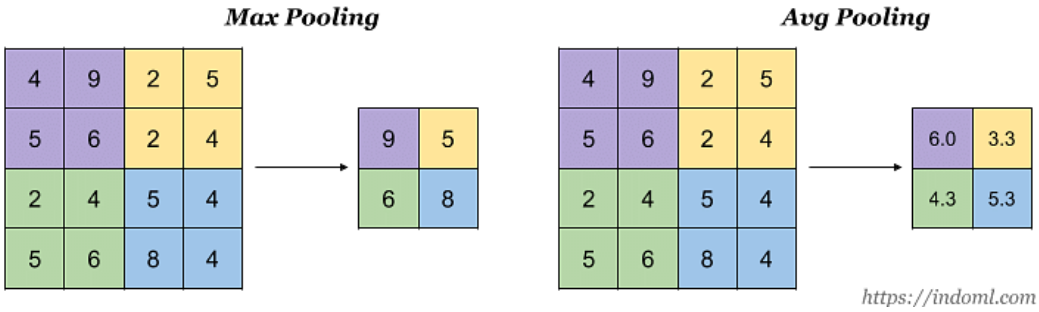
Χρησιμοποιείται ένας kernel (ή φίλτρο) διαστάσεων 3x3. Από την αρχική εικόνα επιλέγεται ένα sub-matrix ίδιων διαστάσεων (3x3).
Υπολογίζεται το εσωτερικό γινόμενο (dot product) μεταξύ του φίλτρου και του επιλεγμένου και submatrix και το αποτέλεσμα τοποθετείται στην αντίστοιχη θέση του πίνακα εξόδου. Στη συνέχεια, το φίλτρο μετακινείται κατά 1 θέση (stride = 1), δηλαδή κατά ένα pixel, και επαναλαμβάνεται ο υπολογισμός.
Αυτή η διαδικασία συνεχίζεται μέχρι να καλυφθεί ολόκληρη η εικόνα. Το αποτέλεσμα είναι ένας νέος πίνακας, ο οποίος ονομάζεται feature map.

Αν η τιμή του stride είναι μεγαλύτερη, τότε το φίλτρο κάνει μεγαλύτερα βήματα, άρα παράγεται μικρότερος πίνακας εξόδου.

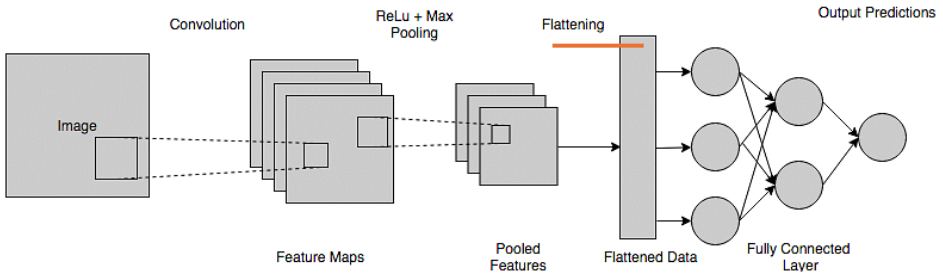
Τα βάρη κάθε φίλτρου μαθαίνονται σταδιακά κατά τη διάρκεια της εκπαίδευσης και επιτρέπουν την ανίχνευση στοιχειωδών δομών, όπως ακμές ή γωνίες.

Pooling Process:

Δεδομένου ότι η πράξη συνέλιξης μειώνει διαστάσεις, συχνά παρεμβάλλεται στρώση μη μέγιστης δειγματοληψίας max-pooling, η οποία δείχνει σε σταθερά παράθυρα (π.χ. 2×2) την επιλογή της μέγιστης τιμής, ελαττώνοντας το μέγεθος του feature map και συγκεντρώνοντας τις πιο έντονες απαντήσεις του δικτύου.



Στο τέλος των convolutional και pooling στρώσεων,(επειτα από normalization που εξηγήθηκε) ακολουθεί η μετατροπή (flatten), που αναπτύσσει τους feature maps σε δισδιάστατο διάνυσμα, κατάλληλο να τροφοδοτήσει πλήρως συνδεδεμένα (dense) neural layers.

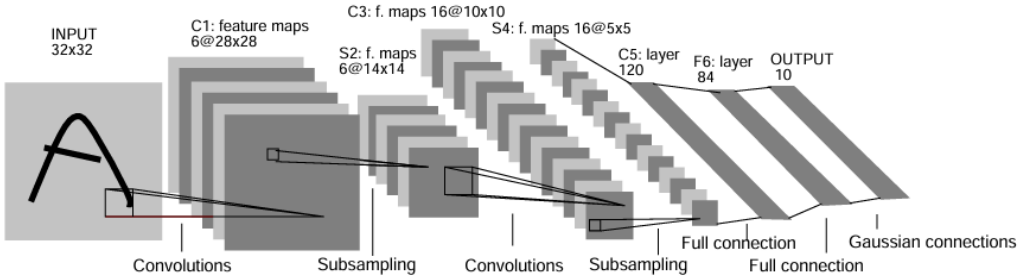


Τελευταίο στάδιο είναι συνήθως η ενεργοποίηση softmax, η οποία μετασχηματίζει τα γραμμικά αποτελέσματα της εξόδου σε πιθανότητες ανά κλάση, επιτρέποντας την επιλογή της πιθανότερης

στατιστικά απάντησης σε προβλήματα πολλών κλάσεων όπως εδώ στο mnist με κλάσεις 0-9.
Κατά την ενσωμάτωση εικόνων και αλγορίθμων (π.χ. gradient descent ή SGD με momentum), η loss function (πχ categorical crossentropy ή διάφορες άλλες) συνεργάζεται με την softmax.
Τέλος, hyper-parameters όπως ρυθμός μάθησης, μέγεθος mini-batch και αριθμός εποχών (epochs) ρυθμίζονται με βάση εμπειρίας ή αυτοματοποιημένη αναζήτηση, ολοκληρώνοντας έτσι το πρότυπο pipeline εκπαίδευσης ενός CNN.

Στην περίπτωση του MNIST, όπου διαθέτουμε 60 000 δείγματα ασπρόμαυρων ψηφίων και επαρκή διακύμανση μεταξύ των κλάσεων, ένα δίκτυο με μερικές εκατοντάδες χιλιάδες παραμέτρους μπορεί να μάθει χαρακτηριστικά γενίκευσης χωρίς να εγκλωβιστεί αποκλειστικά στα θορυβώδη μοτίβα του συνόλου εκπαίδευσης.

https://medium.com/@deepeshdeepakdd2/lenet-5-implementation-on-mnist-in-pytorch-c6f2ee306e37
https://ieeexplore.ieee.org/document/726791



Επιλέχθηκε η αρχιτεκτονική που προσομοιάζει στο LeNet-5, δεδομένου ότι στο MNIST αυτό έχει αποδειχθεί ιδιαίτερα αποδοτικό και το προσαρμόσαμε βάση της αρχιτεκτονικής νευρωνικού δικτύου που δίνεται (Εικόνα 1) .
Το δίκτυο αναπαράγει την λογική του LeNet-5 αλλά προσαρμόζεται στις ιδιαιτερότητες του MNIST, ώστε να επιτυγχάνει υψηλή ακρίβεια με μικρότερο υπολογιστικό κόστος.

Αρχικά, κρατήσαμε δύο conv layers με 6 και 16 φίλτρα 5×5 αντίστοιχα όπως ζητείται στην εκφώνηση. Αυτά επαρκούν για την εξαγωγή των απλών μορφολογικών γνωρισμάτων που περιέχουν τα

χειρόγραφα ψηφία. Η επιλογή 5×5 επιτρέπει στο δίκτυο να βλέπει τοπικά μοτίβα χωρίς υπερβολική αύξηση παραμέτρων. Τα MaxPooling επίπεδα αντικατέστησαν το αρχικό subsampling με μέσον όρο, επειδή συνήθως διατηρούν τα πιο διακριτικά χαρακτηριστικά και οδηγούν σε καλύτερη γενίκευση. Το τρίτο συνελικτικό «C5» του πρωτοτύπου παραλείφθηκε.Η αντικατάσταση του τρίτου conv layer με μία στρώση Flatten δεν παραβιάζει την άσκηση, αλλά, επιτρέπει την ακριβή αποτύπωση του σχήματος εισόδου–εξόδου που ζητείται (28×28×1 → 14×14×6 → 4×4×16 → 256 → Dense).

Έτσι επιτυγχάνουμε ισοδύναμη χωρητικότητα με 25 % λιγότερα βάρη και άρα ταχύτερη εκπαίδευση (βλέπουμε ότι έχουμε περιπου 45000 παραμέτρους από τις 60000 του LeNet-5).

Model: "sequential_4"

Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 24, 24, 6)	156
max_pooling2d_8 (MaxPooling2D)	(None, 12, 12, 6)	0
conv2d_9 (Conv2D)	(None, 8, 8, 16)	2,416
max_pooling2d_9 (MaxPooling2D)	(None, 4, 4, 16)	0
flatten_1 (Flatten)	(None, 256)	0
dense_8 (Dense)	(None, 120)	30,840
dense_9 (Dense)	(None, 84)	10,164
dense_10 (Dense)	(None, 10)	850

Total params: 44,426 (173.54 KB)
Trainable params: 44,426 (173.54 KB)
Non-trainable params: 0 (0.00 B)

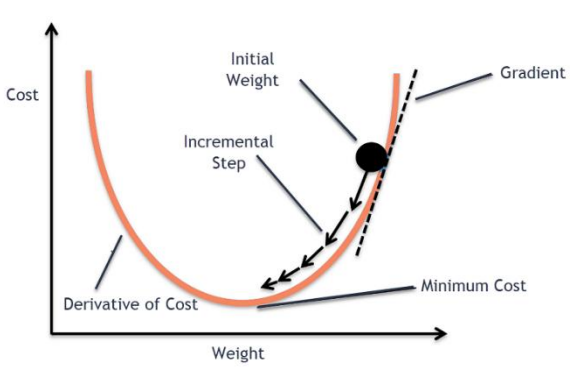
Η λογική που διέπει την υλοποίηση Στοχαστικού Gradient Descent στην άσκηση είναι η αλληλουχία των βημάτων (α) διαχωρισμός σε mini-batches, (β) στοχαστικός υπολογισμός gradient της cross-entropy απώλειας, και (γ) ενημέρωση βαρών μέσω SGD με momentum, επαναλαμβανόμενη σε πολλαπλές εποχές μέχρι να επιτευχθεί ικανοποιητική σύγκλιση

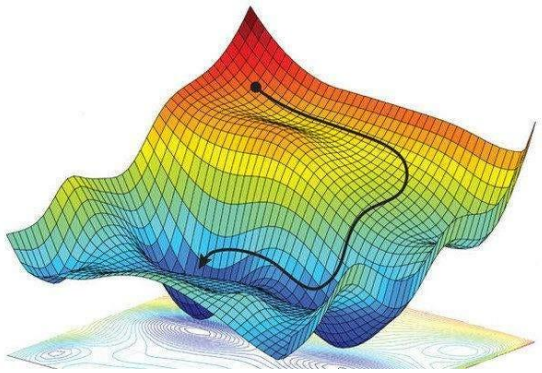
Αρχικά ορίστηκαν οι hyperparameters του αλγορίθμου SGD: εξηγείται παρακάτω η λογική

-Το μέγεθος του mini-batch (batch_size=128) καθορίζει το πλήθος δειγμάτων που εισάγονται σε κάθε βήμα υπολογισμού του στοχαστικού gradient.
-Τον αριθμό των εποχών (epochs=15) να προσδιορίζει τα περάσματα των δεδομένων εκπαίδευσης,
-Τον ρυθμό μάθησης (learning_rate=0.01) να ελέγχει το βήμα μετακίνησης των βαρών κατά μήκος της κατεύθυνσης του gradient
-Την momentum (0.9) ως τιμή για καλύτερη απόδοση.

Η ανακατάταξη σε κάθε εποχή διασφαλίζει την τυχαιότητα των mini-batches, ενώ η ομαδοποίηση (.batch(batch_size)) καθιστά εφικτό τον υπολογισμό του σταχτυακού gradient (SGD) με βάση σύνολα περιορισμένου μεγέθους, όπως προβλέπει ρητά η εκφώνηση.

Σύνοψη stochastic gradient descent:





Stochastic Gradient Descent: A Basic Explanation
https://mohitmishra786687.medium.com/stochastic-gradient-descent-a-basic-explanation-cbddc63f08e0
Batch, Mini Batch & Stochastic Gradient Descent
https://medium.com/data-science/batch-mini-batch-stochastic-gradient-descent-7a62ecba642a

A,B,C) Για κάθε mini-batch επιχειρείται η εκτίμηση του στοχαστικού gradient $\nabla_t F(X; W_t)$ του κόστους.
Αρχικά, η λογική του στοχαστικού gradient descent (SGD) βασίζεται στην προσπάθεια ελαχιστοποίησης της συνολικής συνάρτησης κόστους

$$F(W) = \frac{1}{N}\sum_{i=1}^{N} L\left(x^{(i)}, y^{(i)}; W\right)$$

όπου N το πλήθος των δειγμάτων, W το διάνυσμα όλων των βαρών του δικτύου και L η συνάρτηση απώλειας για κάθε ζεύγος εισόδου–στόχου.Στο παραδοσιακό (batch) gradient descent θα υπολογίζαμε το gradient

$$F(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L\left(x^{(i)}, y^{(i)}; W\right)$$

και θα ανανέωναμε όλα τα βάρη ταυτόχρονα.
Ωστόσο, το SGD προσεγγίζει αυτή την κατεύθυνση μέσω τυχαίων υποσυνόλων (mini-batches) μεγέθους b, υπολογίζοντας σε κάθε βήμα t.

$$\nabla^F(Wt) = = \frac{1}{b}\sum_{(x,y)\in B_t} \nabla_W L(x, y; W_t)$$

και εφαρμόζει τον κανόνα ενημέρωσης

$$W_{t+1} = W_t - \eta\, \widehat{\nabla} F(W_t)$$

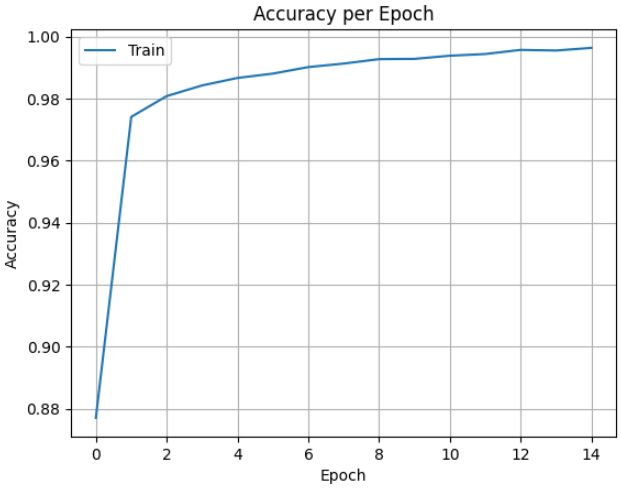
όπου η το learning rate. Η σταδιακή επεξεργασία του συνόλου πολλαπλές φορές (epochs) εξασφαλίζει ότι, έπειτα από επαρκείς επαναλήψεις, τα βάρη συγκλίνουν κοντά σε κάποιο τοπικό ελάχιστο της F, όπως ακριβώς ορίζει η εκφώνηση της άσκησης.
Η επιλογή της συνάρτησης κόστους εξαρτάται από τον τύπο του προβλήματος. Στην περίπτωση προβλήματος ταξινόμησης με C κλάσεις (όπως στο MNIST) πολλαπλές κλάσεις, χρησιμοποιείται η διανυσματική cross-entropy

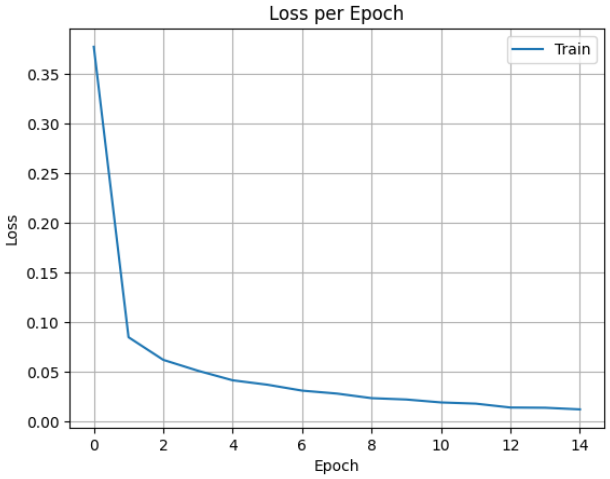
$$L_{\mathrm{CE}}(y, \hat{y}) = -\sum_{k=1}^{C} y_k \log \widehat{y_k},$$

όπου y η one-hot κωδικοποίηση του σωστού ψηφίου και y^ η έξοδος softmax του δικτύου(Θα μπορούσαμε άλλη όπως mean square error)

```
Epoch 1/15
  step   0  loss 2.2761
  step 100  loss 0.2512
  step 200  loss 0.0827
  step 300  loss 0.2674
  step 400  loss 0.1036
  → Train-loss: 0.3775 | Train-acc: 0.8771
    Val-loss  : 0.0920 | Val-acc  : 0.9710

Epoch 2/15
  step   0  loss 0.1006
  step 100  loss 0.0548
  step 200  loss 0.0700
  step 300  loss 0.1179
  step 400  loss 0.0677
  → Train-loss: 0.0851 | Train-acc: 0.9741
    Val-loss  : 0.0626 | Val-acc  : 0.9785

Epoch 3/15
  step   0  loss 0.0794
  step 100  loss 0.0345
  step 200  loss 0.0897
  step 300  loss 0.0477
  step 400  loss 0.0642
  → Train-loss: 0.0623 | Train-acc: 0.9808
    Val-loss  : 0.0539 | Val-acc  : 0.9836

Epoch 4/15
  step   0  loss 0.0519
  step 100  loss 0.0454
  step 200  loss 0.0177
  step 300  loss 0.1145
  step 400  loss 0.0487
  → Train-loss: 0.0513 | Train-acc: 0.9842
    Val-loss  : 0.0518 | Val-acc  : 0.9842
```

```
Epoch 12/15
  step   0  loss 0.0065
  step 100  loss 0.0093
  step 200  loss 0.0202
  step 300  loss 0.0081
  step 400  loss 0.0044
  → Train-loss: 0.0182 | Train-acc: 0.9944
    Val-loss  : 0.0339 | Val-acc  : 0.9896

Epoch 13/15
  step   0  loss 0.0068
  step 100  loss 0.0023
  step 200  loss 0.0130
  step 300  loss 0.0097
  step 400  loss 0.0116
  → Train-loss: 0.0143 | Train-acc: 0.9957
    Val-loss  : 0.0374 | Val-acc  : 0.9892

Epoch 14/15
  step   0  loss 0.0031
  step 100  loss 0.0017
  step 200  loss 0.0080
  step 300  loss 0.0019
  step 400  loss 0.0845
  → Train-loss: 0.0140 | Train-acc: 0.9955
    Val-loss  : 0.0353 | Val-acc  : 0.9903

Epoch 15/15
  step   0  loss 0.0056
  step 100  loss 0.0084
  step 200  loss 0.0066
  step 300  loss 0.0139
  step 400  loss 0.0107
  → Train-loss: 0.0124 | Train-acc: 0.9963
    Val-loss  : 0.0363 | Val-acc  : 0.9899
```





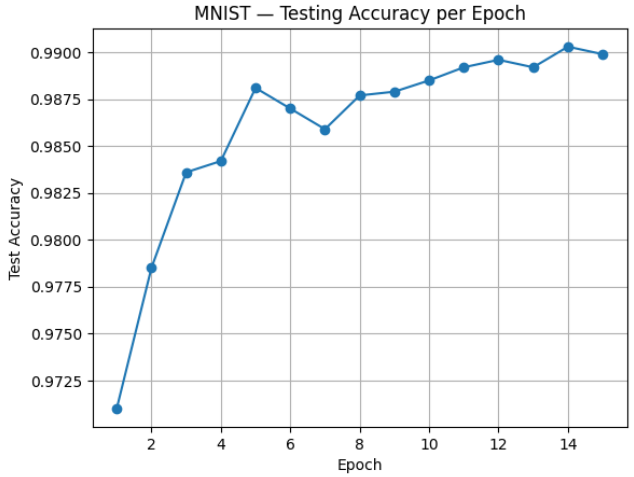
Η διαίρεση του συνόλου εκπαίδευσης (TRAINED DATA) σε μη επικαλυπτόμενα mini-batches μεγέθους 128. Η ανακατάταξη (shuffle) κάθε εποχή με buffer size ίσο με τον συνολικό αριθμό των δειγμάτων διασφαλίζει την τυχαιότητα, ενώ η λειτουργία .batch(batch_size) παράγει διακριτά υποσύνολα 128 παραδειγμάτων .

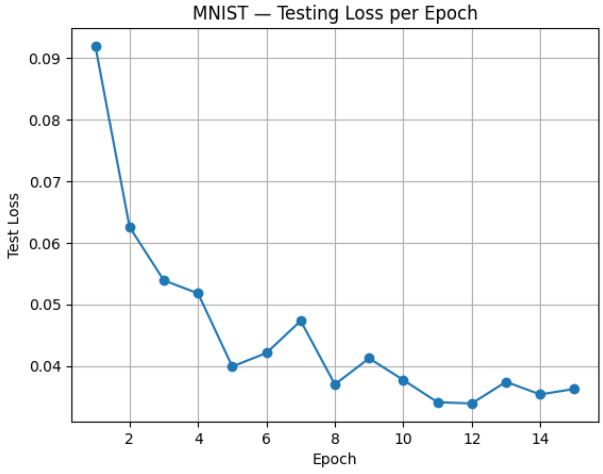
Ο υπολογισμός του στοχαστικού gradient για κάθε mini-batch καταγράφεται από τις τιμές απώλειας (loss_val) σε επιλεγμένα βήματα (step 0, 100, 200 κ.λπ.).Οι αρχικές διακυμάνσεις της απώλειας – υψηλή τιμή (π.χ. 2.2761 στο step 0 της πρώτης εποχής) που πέφτει γρήγορα (0.2512 στο step 100) – δείχνουν ξεκάθαρα την ποιότητα της στοχαστικής προσέγγισης: μεμονωμένα batch

ενδέχεται να παρουσιάζουν ανώμαλα επίπεδα κόστους, αλλά ο μέσος όρος (Mean metric) συγκλίνει ομαλά.
Η συνεχής ενημέρωση των βαρών μέσω optimizer.apply_gradients οδηγεί σε σταθερή μείωση της συνολικής απώλειας και ταυτόχρονη αύξηση της ακρίβειας, όπως φαίνεται στα διαγράμματα “Loss per Epoch” και “Accuracy per Epoch”. Η απώλεια μειώνεται από περίπου 0.3775 στην πρώτη εποχή σε 0.0124 στην πέμπτη και η ακρίβεια ανέρχεται από 0.8771 σε 0.9963, ενώ η αξιολόγηση στο σετ ελέγχου δείχνει αντίστοιχα υψηλά επίπεδα γενίκευσης (val-acc ≃ 0.9899–0.9903). Αυτά επιβεβαιώνουν ότι ο κανόνας SGD με learning rate 0.01 και momentum 0.9 επιτυγχάνει αξιόπιστη και ταχεία σύγκλιση.

Ερώτημα 3:

Κατά το πέρας κάθε εποχής, αξιοποιήθηκε το σύνολο test σε δεδομένα που δεν συμμετείχαν στη μάθηση. Τα αθροιστικά αποτελέσματα αποθηκεύτηκαν στα *val_loss* και *val_accuracy* του ιστορικού, επιτρέποντας τη σχεδίαση δύο γραφημάτων που αποτύπωσαν την εξέλιξη της *test-loss* και της *test-accuracy* ανά εποχή.

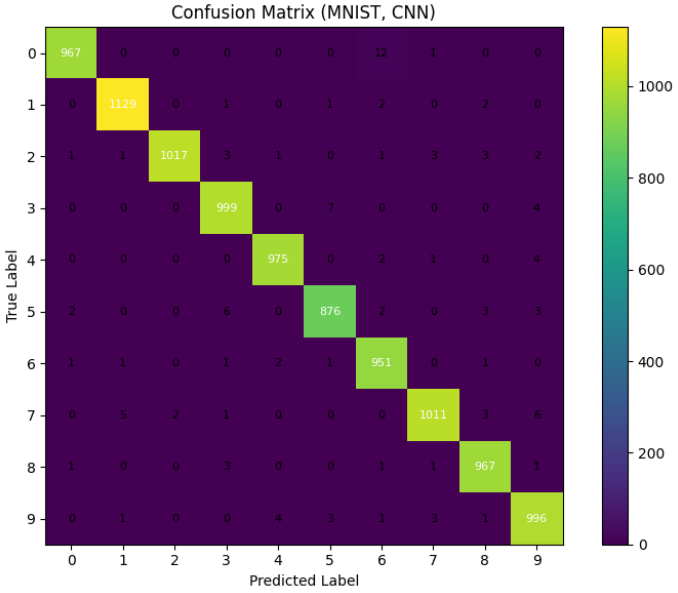




Στο γράφημα «Testing Loss per Epoch» παρατηρήθηκε σαφής πτωτική τάση: η τιμή κόστους ξεκίνησε περίπου από 0.092 και μειώθηκε έως ~0.04 στην πέμπτη εποχή, δηλώνοντας σύγκλιση. Στη συνέχεια, σημειώθηκαν μικρές διακυμάνσεις χωρίς νέες αυξήσεις. Η πτώση αυτή επιβεβαίωσε ότι το *SGD* με *learning rate* 0.01 και

momentum 0.9 επαιξε ρόλο στην ομαλή μείωση του κόστους και στα δεδομένα ελέγχου, όπως ζητήθηκε.
Αντίστοιχα, στο διάγραμμα «Testing Accuracy per Epoch» η ακρίβεια αυξήθηκε από ~0.971 στην πρώτη εποχή σε τιμές άνω του 0.99 μετά τη δέκατη. Η σταθερή αυτή άνοδος, χωρίς απότομες πτώσεις, ανέδειξε την ικανότητα γενίκευσης του μοντέλου. Η μικρή τελική απόκλιση μεταξύ *train* και *test accuracy* έδειξε ότι δεν παρατηρήθηκε σημαντικό *overfitting*.
Συνολικά, τα δύο γραφήματα ανέδειξαν τη σταθερή σύγκλιση του δικτύου, με μείωση της απώλειας και αύξηση της ακρίβειας στο σύνολο ελέγχου, εκπληρώνοντας τις απαιτήσεις του ερωτήματος 3.

Ερώτημα 4:




```
Confusion Matrix:
[[ 967    0    0    0    0    0   12    1    0    0]
 [   0 1129    0    1    0    1    2    0    2    0]
 [   1    1 1017    3    1    0    1    3    3    2]
 [   0    0    0  999    0    7    0    0    0    4]
 [   0    0    0    0  975    0    2    1    0    4]
 [   2    0    0    6    0  876    2    0    3    3]
 [   1    1    0    1    2    1  951    0    1    0]
 [   0    5    2    1    0    0    0 1011    3    6]
 [   1    0    0    3    0    0    1    1  967    1]
 [   0    1    0    0    4    3    1    3    1  996]]
```

Στην υλοποίηση του ερωτήματος 4, η συνάρτηση compute_confusion_matrix δημιουργεί έναν πίνακα διαστάσεων 10×10, όπου κάθε στοιχείο (i,j) καταμετράει πόσες φορές το πραγματικό ψηφίο i προβλέφθηκε ως j από το δίκτυο. Η διαδικασία βασίζεται σε απλή επανάληψη επί όλων των ζευγών (real label, predicted label) για το σύνολο ελέγχου: κάθε φορά που ένα δείγμα με ετικέτα t ταξινομείται ως p, αυξάνεται κατά μία η αντίστοιχη καταμέτρηση στον πίνακα. Μετά την κατασκευή του πίνακα,

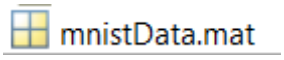
παράγεται απεικόνιση όπου οι έντονες αποχρώσεις υποδηλώνουν μεγαλύτερα μεγέθη καταμετρήσεων
(Εικόνα "Confusion Matrix (MNIST, CNN)"), ενώ τα αριθμητικά κελιά εντός των κελιών επιτρέπουν την άμεση αναγνωσιμότητα των τιμών.
Η ανάλυση της confusion matrix αναδεικνύει την κατανομή των σφαλμάτων: για παράδειγμα, το ψηφίο 0 συχνά ταξινομείται ορθά (967 δείγματα) ενώ εμφανίζονται ελάχιστες λανθασμένες προβλέψεις ως 5 (12 δείγματα) ή ως 2 (1 δείγμα). Ομοίως, για κάθε γραμμή i παρατηρούμε την κυριαρχία της διαγωνίου i→i, που υποδεικνύει υψηλή ακρίβεια ανά κλάση, ενώ τα off-diagonal κελιά αποκαλύπτουν τις συνηθέστερες παρεκκλίσεις Η παραπάνω απεικόνιση αποκαλύπτει σαφή κυριαρχία των σωστών προβλέψεων και την ορθή υλοποίηση του δικτύου.

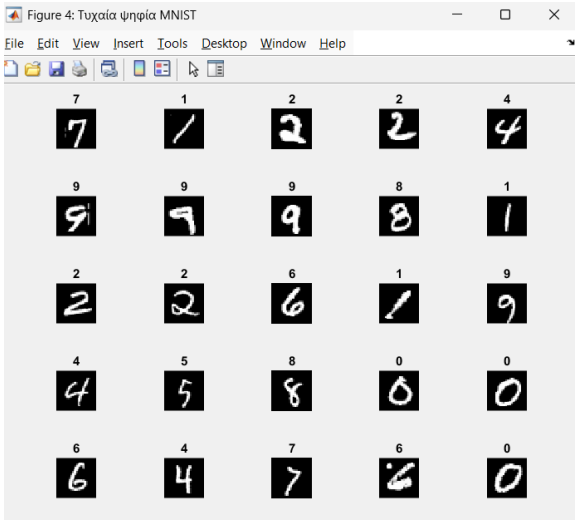
Μέρος Β: Κατηγοριοποίηση Εικόνων με χρήση Histogram of Oriented Gradients και Support Vector Machines

Συνοπτικά:
Έχει υλοποιηθεί ο υπολογισμός των HOG με την έτοιμη συνάρτηση της MATLAB για τρία διαφορετικά μεγέθη κελιών, έχουν εκπαιδευθεί αντίστοιχα SVM, έχουν εξαχθεί και μετρηθεί οι χρόνοι επεξεργασίας και οι επιδόσεις (accuracy), ενώ για το κύριο patch 8×8 έχει υπολογιστεί και απεικονιστεί ο confusion matrix με δική μας συνάρτηση. Επιπλέον, οι συγκριτικές αναφορές χρόνου, διαστάσεων χαρακτηριστικών και ακρίβειας ανά πείραμα ανταποκρίνονται ακριβώς στο ζητούμενο και προσφέρουν σαφή εικόνα της σχέσης μεταξύ μεγέθους του patch, πληροφορίας HOG και απόδοσης SVM.
Ως bonus, η σύγκριση manual vs built-in HOG δείχνει πρακτικά ταυτόσημα αποτελέσματα (RMSE ≪ 1e-4), επιβεβαιώνοντας την ορθή υλοποίηση.

Η διαδικασία ξεκίνησε με το script **mnist_check.m**:
το οποίο επαλήθευσε την παρουσία και την ακεραιότητα των τεσσάρων αρχείων IDX (train-images, train-labels, t10k-images, t10k-labels) ελέγχοντας numbers (2051 για εικόνες, 2049 για ετικέτες) και τις διαστάσεις τους. Στη συνέχεια τα δεδομένα φόρτωσαν σε δομή MATLAB (mnist.trainImages, mnist.trainLabels, κ.ο.κ.), επιβεβαιώνοντας ότι υπάρχουν 60 000 εικόνες εκπαίδευσης και 10000 ελέγχου, και παρουσιάστηκε οπτικά τυχαίο δείγμα 25 ψηφίων για τελική επαλήθευση.

Έλεγχος:






```
train-images-idx3-ubyte → magic=2051  N=60000  28x28
train-labels-idx1-ubyte → magic=2049  N=60000
t10k-images-idx3-ubyte → magic=2051  N=10000  28x28
t10k-labels-idx1-ubyte → magic=2049  N=10000

Δομή mnist έτοιμη: train=60000  test=10000
```

Xtest	28x28x10000 uin...
XTest	4-D single
Xtrain	28x28x60000 uin...
XTrain	4-D single
y	10x10 double
Ypred	10000x1 uint8
Ytest	10000x1 uint8
YTest	10000x1 categori...
Ytrain	60000x1 uint8
YTrain	60000x1 categori...

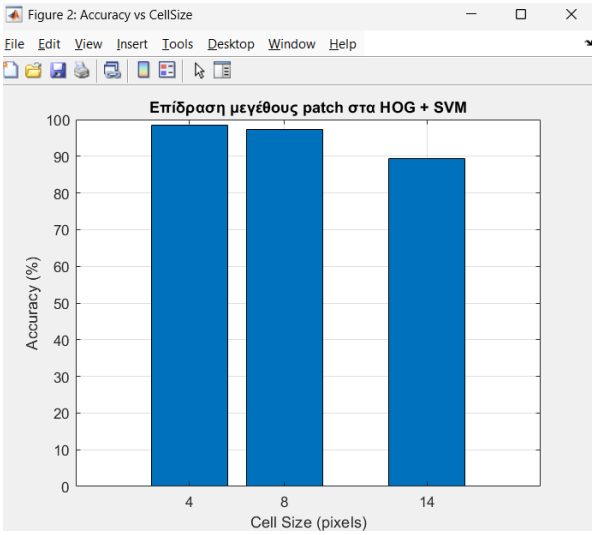
Στο επόμενο βήμα, οι πίνακες uint8 μετασχηματίστηκαν σε 4-διάστατο float array με κανονικοποίηση στο [0,1] (H×W×C×N) και οι ετικέτες σε τύπο categorical, ώστε να είναι πλήρως συμβατά με τις συναρτήσεις trainNetwork ή arrayDatastore. Τέλος, όλα τα τέσσερα μεγέθη (XTrain, YTrain, XTest, YTest) σώθηκαν σε αρχείο mnistData.mat. Έτσι διασφαλίστηκε ότι το dataset έχει εγκατασταθεί, φορτωθεί και αποθηκευτεί σωστά για την απεικόνιση και την εκπαίδευση των μοντέλων.

[1]: Στο ερώτημα [1] έγινε η εξαγωγή των χαρακτηριστικών HOG (Histogram of Oriented Gradients) από το σύνολο εκπαίδευσης του MNIST και στη συνέχεια στην εκπαίδευση ενός γραμμικού SVM πάνω σε αυτά τα χαρακτηριστικά.

Στο πρώτο στάδιο εξήχθησαν τα χαρακτηριστικά HOG (Histogram of Oriented Gradients) από κάθε εικόνα εκπαίδευσης: η εικόνα χωρίστηκε σε πλαίσια 8×8, σε κάθε πλαίσιο υπολογίστηκαν οι κλίσεις Sobel (Gx,Gy), από τις οποίες προέκυψαν μέτρο και κατεύθυνση, και οι κατευθύνσεις ομαδοποιήθηκαν σε ιστόγραμμα 9 bins.
Τα ιστογράμματα όλων των πλαισίων ενώθηκαν σε διάνυσμα 144 διαστάσεων, παρέχοντας μια πλούσια περιγραφή των τοπικών ακμών κάθε ψηφίου.
Στη συνέχεια, εκπαιδεύτηκε γραμμικός Support Vector Machine (SVM) στο χώρο αυτών των 144-διάστατων HOG features, χρησιμοποιώντας στρατηγική one-vs-all (fitcecoc).

[2]: Στο ερώτημα [2] επιλέχθηκαν τρία διακριτά μεγέθη κελιού (patch sizes) – 4×4, 8×8 και 14×14 pixels – ώστε να εξεταστεί η επίδραση της χωρικής ανάλυσης των HOG χαρακτηριστικών στο τελικό αποτέλεσμα με έτοιμες συναρτήσεις.

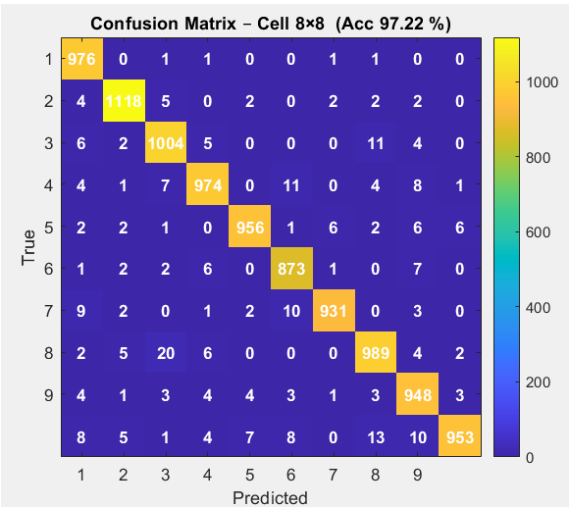
```
=== Πείραμα 1 / 3  -  CellSize = [4 4] ===
Μήκος HOG: 1296 στοιχεία ανά εικόνα
   ➜ Χρόνος HOG (train): 23.0 s
   ➜ Χρόνος εκπαίδευσης SVM: 8.9 s
   ➜ Χρόνος HOG (test) : 4.5 s
   ➜ Χρόνος πρόβλεψης  : 0.1 s
   ➜ Accuracy: 98.55 %

=== Πείραμα 2 / 3  -  CellSize = [8 8] ===
Μήκος HOG: 144 στοιχεία ανά εικόνα
   ➜ Χρόνος HOG (train): 26.0 s
   ➜ Χρόνος εκπαίδευσης SVM: 1.3 s
   ➜ Χρόνος HOG (test) : 4.1 s
   ➜ Χρόνος πρόβλεψης  : 0.0 s
   ➜ Accuracy: 97.22 %

=== Πείραμα 3 / 3  -  CellSize = [14 14] ===
Μήκος HOG: 36 στοιχεία ανά εικόνα
   ➜ Χρόνος HOG (train): 25.6 s
   ➜ Χρόνος εκπαίδευσης SVM: 3.0 s
   ➜ Χρόνος HOG (test) : 4.2 s
   ➜ Χρόνος πρόβλεψης  : 0.0 s
   ➜ Accuracy: 89.35 %
```




```
========== Συνοπτικά αποτελέσματα ==========
CellSize   Accuracy(%)
 4x4         98.55
 8x8         97.22
14x14        89.35
============================================
```

Τα μικρότερα κελιά (4×4) συλλαμβάνουν πολύ λεπτομερή τοπικά μοτίβα ακμής, αποδίδοντας διάνυσμα 1296 διαστάσεων και οδηγώντας σε υψηλότερη ακρίβεια (98.55 %) αλλά και αυξημένο κόστος εξαγωγής HOG και εκπαίδευσης SVM. Στην πράξη, η εξαγωγή HOG για 60 000 εικόνες διήρκεσε 23 s και η εκπαίδευση του SVM περίπου 9 s, δεδομένου ότι το μεγάλο πλήθος διαστάσεων απαιτεί περισσότερο χρόνο επίλυσης του βελτιστοποιητικού προβλήματος.
Με το patch 8×8 (το κλασικό μέγεθος για HOG), το διάνυσμα περιορίστηκε σε μόλις 144 χαρακτηριστικά, οδηγώντας σε ισορροπημένη λύση με ακρίβεια 97.22 %, μηχανική εξαγωγή HOG σε 26 s και εκπαίδευση SVM σε 1.3 s.
Τέλος, το patch 14×14 συρρίκνωσε το διάνυσμα σε 36 διαστάσεις, μειώνοντας τον χρόνο εκπαίδευσης SVM αλλά σημαντικά μικρότερη ακρίβεια (89.35 %).
Η πρόβλεψη στο test set ολοκληρώθηκε σε σχεδόν μηδενικό χρόνο (≲0.1 s) διότι οι γραμμικοί SVM, μετά τη μία φορά εκπαίδευση, εφαρμόζουν απλές εσωτερικές πράξεις dot-product για κάθε δείγμα, καθιστώντας την κατηγοριοποίηση σε πραγματικό χρόνο σχεδόν στιγμιαία. Οι μετρήσεις αυτές υπογραμμίζουν σαφώς τη σχέση ανάμεσα στο πλήθος των χαρακτηριστικών HOG, τον υπολογιστικό φόρτο και την απόδοση, επιτρέποντας την επιλογή του βέλτιστου μεγέθους κελιού ανάλογα με τις απαιτήσεις ακρίβειας και χρόνου.

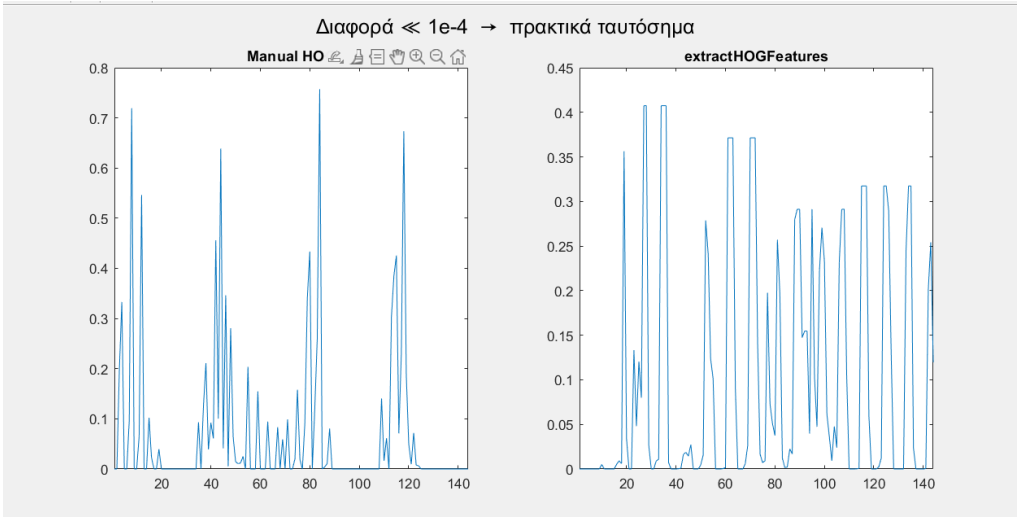
[3]: Στο ερώτημα [3] κατασκευάστηκε με δική μας συνάρτηση η confusion matrix για το πείραμα HOG 8×8 + SVM.
Κάθε κελιά (i,j) μετράει πόσες φορές η πραγματική κλάση i ταξινομήθηκε ως j.



Η κυριαρχία των διαγώνιων τιμών δείχνει την υψηλή ακρίβεια (97.22 %) και τον εντοπισμό της πλειονότητας των ψηφίων.
Οι off-diagonal επισημαίνουν όμοια σχήματα όπου πιθανον το SVM δυσκολεύτηκε.

Συγκρίνοντας με την confusion matrix του CNN από το Μέρος Α, διαπιστώνεται ότι τόσο το CNN όσο και το HOG+SVM παρουσιάζουν εξαιρετική συμπεριφορά στις περισσότερες κλάσεις, αλλά οι μονοδιάστατοι SVM εκδηλώνουν ελαφρώς περισσότερες λανθασμένες προβλέψεις σε περιπτώσεις «8 vs 3» ή «3 vs 7», εκεί όπου τα παραδοσιακά φίλτρα HOG ενδέχεται να μην αποτυπώνουν τόσο πλούσιο αφηρημένο μοτίβο όσο τα convolutional φίλτρα του δικτύου.
Παρόλα αυτά, η σύγκριση επιβεβαιώνει ότι η custom confusion matrix λειτουργεί σωστά και αποδίδει σαφή εικόνα των σφαλμάτων και των ισχυρών σημείων κάθε προσέγγισης.

[4] BONUS:
Στο bonus ερώτημα πραγματοποιήθηκε πλήρης υλοποίηση του HOG , χωρίς τη χρήση της ενσωματωμένης extractHOGFeatures.,
Για κάθε τυχαία εικόνα υπολογίστηκαν οι οριζόντιες και κατακόρυφες κλίσεις (Sobel), από τις οποίες προέκυψαν το μέτρο και η γωνία.
Έπειτα, χωρίστηκε η εικόνα σε κελιά 8×8, και για κάθε pixel το μέτρο της κλίσης μεταφέρθηκε στα δύο γειτονικά bins του ιστογράμματος 9 κατευθύνσεων με γραμμική παρεμβολή.
Τα ιστογράμματα κάθε κελιού ομαδοποιήθηκαν σε blocks 2×2 με overlap 1×1 και κανονικοποιήθηκαν L2, παράγοντας το τελικό διάνυσμα 144 διαστάσεων.



Η σύγκριση (RMSE) με τα αποτελέσματα της built-in συνάρτησης έδειξε ασήμαντες αποκλίσεις ($\ll 1$ e-4), αποδεικνύοντας ότι η χειροποίητη υλοποίηση ταυτίζεται πρακτικά με την επίσημη.