Neural Networks & Deep Learning
Φωτεινή Κοκκώνη

Επιλογή Dataset

Επέλεξα την Β επιλογή, μετά από πολύ μελέτη και εξερεύνηση των dataset καθώς και αντιμετώπιση προβλημάτων αποφάσισα να χρησιμοποιήσω το dataset:

www.kaggle.com/datasets/chriskjm/polish-traffic-signs-dataset}, το οποίο και κατέβασα στον υπολογιστή μου.

Read και Διαμόρφωση των Data

Αρχικά έπρεπε να φορτώσω τα images και τα labels του dataset μου.Όρισα 2 lists,ένα για τα data που θα δίαβασω και ένα για debugging.Με χρήση της βιβλιοθήκης os κατάφερα να κάνω το πρόγραμμα μου να ακολουθεί τα path στο directory μου.Το συγκεκριμένο dataset είχε δικό του, έτοιμο, classification. Μέσα στον φάκελο classification υπάρχουν τα αντίστοιχα folders(κλάσεις) τις οποίες και κάνω read αλλά και append στο data_list μου(ουσιαστικά οι εικόνες μου λειτουργούν ως τα labels μου).Με το append όπως το έχω γράψει ουσιαστικά δημιουργώ ένα dictionary με τα βασικά στοιχεία των κλάσεων μου όπως filename (στην περίπτωση μου A1,A17,A2 κτλ),το path και βέβαια το όνομα της κλάσης όπως π.χ το B22 αντιστοιχεί στις πινακίδες ταχύτητας.Μετά χρησιμοποιώ μία βιβλιοθήκη που μετατρέπει το data_list μου σε ένα 2d πινακάκι ουσιαστικά όπως έχουμε δει και στην sql. Ακολουθεί debugging message για να δω ότι τα πάντα λειτουργούν σωστά. Ακολουθεί ένα class_mapping που απλά δημιουργώ αριθμητικά labels για το πινακάκι μου και ακολουθεί printing για επιβεβαίωση των labels.

Επίσης οι κλάσεις μου είναι οι εξής:

A1=Απότομη στροφή δεξιά	A30=Προσοχή Κίνδυνος	B22=Απαγορεύεται η στροφή δεξιά	C2=Υποχρεωτική στροφή δεξιά	Other=Random signs
A2= Απότομη στροφή αριστερά	B1=Κλειστή οδός για τα οχήματα και από τις 2 κατ.	B23=Απαγορεύεται η αναστροφή	C4=Υποχρεωτική στροφή αριστερά	
A7=Προτεραιότητα	B2=Απαγορεύεται η είσοδος σε όλα τα οχήματα	B33=Όριο Ταχύτητας	C12=Κυκλική υποχρεωτική διαδρομή	
A17=Προσοχή Πεζοί	B20=ΣΤΟΠ	B36=Απαγορεύεται η στάση και η στάθμευση	D1=Οδός προτεραιότητας	
A21=Προσοχή διέλευση τραμ	B21=Απαγορεύεται η στροφή αριστερά	B41=Απαγορεύεται η είσοδος στους πεζούς	D6=Διάβαση πεζών	

Resizing, Flattening n PCA

Αρχικά μετά από ψάξιμο και αρκετές αποτυχημένες προσπάθειες κατάλαβα ότι οι εικόνες μου έχουν διαφορετικά μεγέθη…έτσι δημιούργησα ένα X_data_list και με ένα for έκανα τις εικόνες μου resize και τις μετέτρεψα σε grayscale για να απλοποιήσω την διαδικασία καθώς το χρώμα χρειάζεται RGB {red%,green%,black%} ενώ στον grayscale έχω μόνο brightness{0=total black – 255 =white}.Έπειτα τα κάνω append στην λίστα μου ενώ χρησιμοποιώ και την συνάρτηση flatten για να δημιουργήσω έναν 1d vector διότι είναι βασικό για το επόμενο βήμα με PCA.Το PCA το χρησιμοποιώ για να συμπιέσω τις διαστάσεις μου καθώς οι αρχικές είναι 32*32=1024 pixels και μετά το PCA γίνονται 50 οι διαστάσεις με τις οποίες πρέπει να μπλέξουμε.

Train/Testing n Classifiers

Χωρίζω τα data μου σε X_train,X_test που είναι για training και testing αντίστοιχα και τα y_train,y_testing αποτελούν τις εξόδους τους αντίστοιχα X_data_reduced είναι ένας πίνακας με κάθε σειρά να αποτελεί μιας εικόνας «συμπιεσμένα χαρακτηριστικά»(αυτό που ανέφερα από πάνω για τις διαστάσεις),y_data=η συστοιχία των ετικετών.Test_size=0.4 όπως μας ζητήθηκε και το random_state=42 το πήρα από το internet έτοιμο,ουσιαστικά φροντίζει ότι κάθε φορά που θα τρέξεις το πρόγραμμα θα πάρεις το ίδιο split στο data.Τέλος το stratify=y_data είναι για να σιγουρευτούμε ότι κάθε κλάση θα «εμφανίζεται» σε ίσα ποσοστά τόσο στο training όσο και στο testing.
Θέτω τους classifiers μου(Knn k=1,Knn k=3 και Centroid).Δημιουργώ έναν πίνακα για τα αποτελέσματα που θα πάρω και τους έτοιμους classifiers από την βιβλιοθήκη sklearn και θα τους τρέξω.

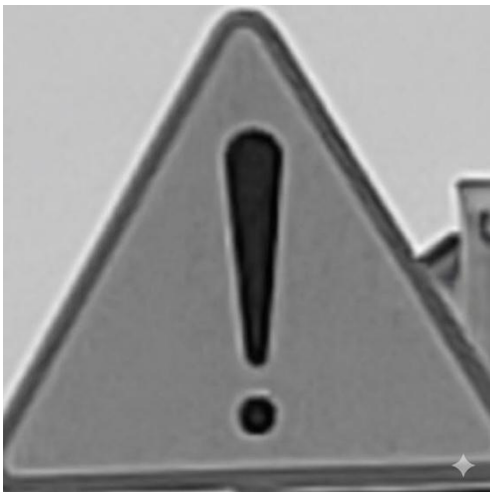
Processing n Accuracy

Τέλος τρέχω τους αλγόριθμους και παίρνω τα παρακάτω αποτελέσματα:
1-NN=0.7871
2-NN=0.7644
N-Centroid=0.4454
Για περισσότερες πληροφορίες του κώδικα υπάρχουν comments πάνω/δίπλα στις εντολές. Υπήρχε επίσης ένα πιο αναλυτικό report αλλά το έβαλα σε comments.
Παρατηρώ ότι ο k=1 είναι ο καλύτερος και πολύ κοντινά στατιστικά έχει και ο k=3 ενώ ο Centroid έχει ΠΟΛΥ χαμηλό accuracy.Γιατί γίνεται αυτό?Ο κ=1 αποτυπώνει τέλεια τα τοπικά μοτίβα και διακρίνει εύκολα images μου είναι δύσκολα αντιληπτά και λόγω του memorise που κάνει γι'αυτό έχει και το πιο υψηλό accuracy.Ο Centroid έχει κάκιστο accuracy καθώς υπεραπλουστεύει τις κλάσεις και στο συγκεκριμένο dataset με τα traffic signs που έχουμε πολλά παρόμοια σχήματα και πινακίδες δεν είναι καθόλου καλό.

BONUS:
Το αρχείο με όνομα KNN εμπεριέχει την προσπάθεια μου να γράψω τους Knn1(k=1),Knn3(k=3) κ Centroid από την αρχή.Τους έτρεξα επίσης στο KYPIO αρχείο που εμπεριέχει και τους έτοιμους από sklearn και πήρα τα ίδια αποτελέσματα,οπότε υποθέτω είναι σωστοί.Υπάρχουν comment στον κώδικα για τα commands ώστε να δείτε το σκεπτικό μου.

MLP n Neurons

Λόγο της μικρής βάσης δεδομένων που επέλεξα αλλά και των σχετικά απλών εικόνων {traffic signs} αποφάσισα να χρησιμοποιήσω ένα MLP μοντέλο(παρόλο που πολύ αργότερα κάναμε την θεωρία για τα cnn και δοκίμασα να τρέξω σε cnn και πήρα καλύτερες αποδώσει).Είχα μία supervised learning προσέγγιση, αρχικά έκανα read τα δεδομένα μου και τα μετέτρεψα σε grayscale{ασπρόμαυρο καθώς έχω RGB και θέλω να μειώσω τα χαρακτηριστικά για απλοποίηση του προβλήματός μου} .Όπως αναφέρθηκα και παραπάνω κάνω resize 32x32=1024 και έπειτα flattening σε 1D πίνακα καθώς έπειτα κάνω το mapping όπου δημιουργώ αριθμητικά labels που αντιστοιχούν σε κάθε class μου.PCA είναι ουσιαστικά μια τεχνική που ΑΠΟΜΟΝΩΝΕΙ τις βασικές πληροφορίες μία εικόνας και «πετάει» τις άχρηστες. Παραδείγματος χάριν σε μία τέτοια εικόνα αφού <μελετήσει> όλα τα δείγματα παρατηρεί το κοινό pattern και τα κύρια χαρακτηριστικά αλλά και την συσχετικότητας τους.Στην εικόνα μας το κίτρινο έχει υψηλή συσχέτιση με τον γείτονα του που είναι το μαύρο ! σύμβολο αλλά και με τα όρια της πινακίδας που έχουν κόκκινο χρώμα.Έτσι καταλαβαίνει κανείς ότι ένα στοιχείο όπως το background το οποίο αλλάζει αλλά και δεν έχει κάποια δυνατή συσχέτιση δεν αποτελεί σημαντικό στοιχείο του pattern recognition.Επίσης το PCA δουλεύει με διακύμανση, δηλαδή που έχει να κάνει με τις μεγάλες αλλαγές στις τιμές των pixel όπως παραδείγματος χάρη το κόκκινο όριο {σε grayscale} με το background{σε grayscale} έχουν υψηλή διακύμανση.



Όρα να τρέξουμε το MLP μας, αρχικά γνωρίζουμε ότι 1 neuron λαμβάνει το άθροισμα των βαρών των προηγούμενων νευρώνων, το input πολλαπλασιάζεται με το w{weight} και προστίθεται ένας όρος μεροληψίας όπως στον παρακάτω τύπο.

$$\text{Net Input} = \left(\sum_{i=1}^{N_{in}} w_i x_i\right) + b$$

Επίσης χρειάζεται και μία συνάρτηση ενεργοποίησης που στην δική μας περίπτωση είναι η 'relu'

$$\text{Output} = f(\text{Net Input})$$

Στο πρώτο πείραμα χρησιμοποιήθηκαν 50 input neurons και 100 hidden_layer neurons στην relu.Ο κάθε νευρώνας στο hidden layer κάνει την πράξη του NetInput και έπειτα χρησιμοποιεί την Relu(f(x)=max(0,x)) συνάρτηση ενεργοποίησης. Με αυτόν τον τρόπο αυτό το layer δημιουργεί μία νέα και «εκπαιδευμένη» αναπαράσταση του input μας για το επόμενο layer να αναλύσει. Έπειτα έχουμε το output layer που παράγουν τα τελικά αποτελέσματα του classification μας.

Back-Propagation

Τι είναι το back-propagation?Είναι μία επαναληπτική διαδικασία πολλαπλών βημάτων που περιλαμβάνει ένας πέρασμα προς τα μπρος και ένα προ τα πίσω. Ο Adam solver που χρησιμοποιούμε έχει ως επίκεντρό του την διαδικασία του back-propagation ώστε να εκπαιδεύσει με ευεξία το MLP μοντέλο μας με στόχο του να ρυθμίζει τα βάρη και τον όρο μεροληψίας ώστε να ελαχιστοποιεί το prediction error(aka loss)

Forward-Pass:
Η διαδικασία που εξηγήθηκε και πριν με τα layers και την επεξεργασία που γίνεται σε κάθε νευρώνα καθώς και το πέρασμα προς τα μπρος στο επόμενο επίπεδο.

Error Calculation:
Είναι μία συνάρτηση που ουσιαστικά συγκρίνει το prediction του MLP (^Y) με την πραγματική ετικέτα(Ytrain) για να υπολογίσει το λάθος(loss).Αυτό αποτελεί και ένα μέτρο του πόσο «κακή» είναι η πρόβλεψη μας.

Backward Pass ή αλλιώς Gradient Calculation:
Backward Pass ροή : output→hidden→input
Η κλίση {gradient} καθορίζει το πόσο και πως πρέπει να αλλάξουν τα βάρη για να μειωθεί το loss καθώς μας δίνει μία κατεύθυνση αλλά και το «μέγεθος» της αλλαγής που πρέπει να γίνει.
Weight Update:
Με τον αλγόριθμο Adam έχουμε :

$$\text{New Weight} = \text{Old Weight} - (\text{Learning Rate} \times \text{Gradient})$$

Όπου ουσιαστικά τα βάρη ανανεώνονται επανειλημμένα με κάθε data batch εκπαίδευσης για το συγκεκριμένο νούμερο «εποχών»(epochs) που έχουμε ορίσει, στη περίπτωση μας max_iter=300

Statistics&Experiment

1.Fist Run:

Έχουμε για αρχή στατιστικά:

	Accuracy(Test)	Accuracy(Train)	Train Time	Test Time
1-NN	**0.7871**	**1.0000**	**0.0000**	**0.2497**
3-NN	**0.7605**	**0.8856**	**0.0000**	**0.2625**
Centroid	**0.4454**	**0.4555**	**0.0005**	**0.0680**
Neural Network(MLP)	**0.8970**	**1.0000**	**1.7796**	**0.0069**

Παρατηρώ ότι το Training_accuracy είναι μεγαλύτερο από το Test_accuracy το οποίο σημαίνει ότι αντιμετωπίζω πρόβλημα overfitting.What is overfitting?
Το μοντέλο τα πάει πάρα πολύ καλά στο training data αλλά όχι στο testing data.
Αυτό γιατί μπορεί να συμβαίνει?

Πιθανές αιτίες:
1.Πολύ περίπλοκο Μοντέλο/Κώδικας/Υλοποίηση
2.Πιθανή Έλλειψη αρκετού data
3.Πολλές περίοδοι training{epochs}
4.Πιθανή Έλλειψη Ομαλοποίησης

Τώρα θα τρέξουμε διάφορα πειράματα για δοκιμή αλλά και παρατήρηση της λειτουργίας του MLP μας καθώς και της επίδοσής του.

2.Experiments

2.1)**Αλλαγή του αριθμού των νευρώνων μας στο κρυφό επίπεδο από 100 σε 50:**
Test time μειώνεται το Train time μειώνεται αλλά το Train accuracy παραμένει σταθερό καθώς και το Test accuracy μειώνεται .Δεν βλέπουμε κάποια ειδοποιός διαφορά για λύση του προβλήματος overfitting μας ούτε κάποιο σημαντικό improvement.

2.2)**Αλλαγή του epoch:**
2.2.1)**Μείωση του epoch(300→100):**
Παρατηρούμε μείωση του Train και Test time καθώς και αρκετή μείωση του Train Accuracy και μία πιο μικρή μείωση του Test Accuracy.Στην συνέχεια μείωσα και άλλο τα epoch αλλά δεν είχαμε καλυτέρευση των αποτελεσμάτων μου
2.2.2)**Αύξηση του epoch(300→500):**
Δεν είχε κανένα καλό αποτέλεσμα και impact άρα δεν υπάρχει λόγος καταγραφής των αποτελεσμάτων.

2.3)**Εισαγωγή του alpha "regulator"(Ουσιαστικά τιμωρεί τα πολύ «μεγάλα» βάρη)**

Με την εισαγωγή του alpha για την ομαλοποίηση αλλάζουμε ουσιαστικά την συνάρτηση loss:

$$L = L_{\text{data}} + \alpha \sum_i w_i^2$$

2.3.1)**alpha[0.001]:**

Μόνο με την εισαγωγή μόνο του alpha δεν βλέπουμε κάποια αισθητή βελτίωση ΟΜΩΣ τα πάντα αλλάζουν με την εισαγωγή early_stopping και n_iten_no_change. Με αlpha=0.001,epoch=300 και 100 hidden layer neurons έχουμε:

	Accuracy(Test)	**Accuracy(Train)**	**Train Time**	**Test Time**
MLP	**0.8432**	**0.9215**	**0.2508**	**0.0034**

Βλέπουμε μια 7.85% μείωση του Train Accuracy και μια 5.38% μείωση στο Test Accuracy.Πάμε να δούμε άμα μπορούμε να το αυξήσουμε με αλλαγή του alpha

2.3.1)**alpa(0.001→0.1)(Πέρασα το 0.01 διότι δεν κάνει κάποια τεράστια διαφορά)**

Με αυτή την αλλαγή είχα πολύ καλύτερα αποτελέσματα καθώς το Test Accuracy ανέβηκε στα 0.8557 και το Training Accuracy στο 0.9293 .Οπότε είχαμε μια 1.25% αύξηση στο Test Ac ενώ στο Training Ac μονάχα 0.73%.

2.4)**Data check**

Σε αυτό το πείραμα τα μειώσω το αρχικό μου database στο μισό του, θα τρέξω τον κώδικα και θα κάνω σύγκριση των αποτελεσμάτων για να καταλάβω άμα απλώς για το overfitting επίσης ευθύνεται η έλλειψη data.

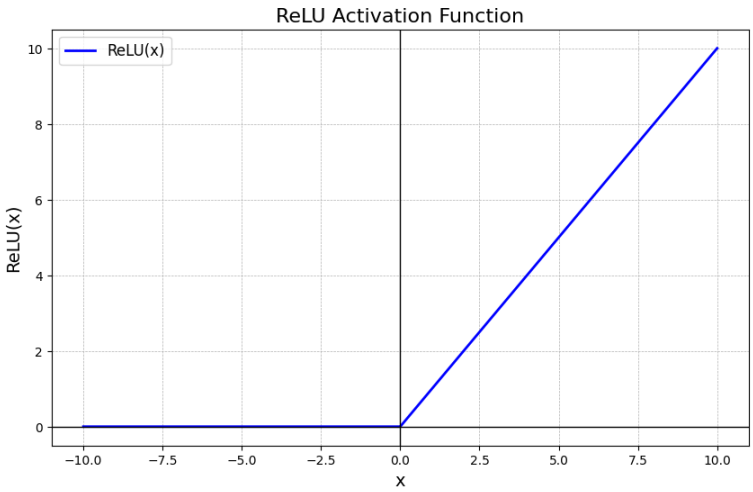
	Accuracy(Test)	**Accuracy(Train)**	**Train Time**	**Test Time**
MLP	**0.8303**	**0.9202**	**0.2594**	**0.0006**

Στο τελικό αποτέλεσμα του 2.3.1 είχαμε διαφορά Train-Test=0.0736 ενώ εδώ σύμφωνα με τις παραπάνω μετρήσεις μας είναι στα 0.0899 που και βέβαια είναι χειρότερο από την περίπτωση 2.3.1.Έτσι υπάρχει πιθανότητα απλός η βάση δεδομένων μας να είναι ελλιπείς και να χρειαζόμαστε μεγαλύτερο όγκο δεδομένων.

2.5)Με αύξηση των νευρώνων πήρα την καλύτερη διαφορά Training-Test {accuracy} με 200 νευρώνες στα 0.0681.Άρα λογικά και με μεγαλύτερο dataset αυτή η διαφορά θα μειωθεί ακόμα περισσότερο. ΟΜΩΣ από τους 300 νευρώνες και μετά τα αποτελέσματα γίνονται χειρότερα με αποτέλεσμα το peak μας να είναι στους 300.

2.6)Πίνακας αποτελεσμάτων «παίζοντας» με adam solver +1 hidden layer

Relu Activation Function



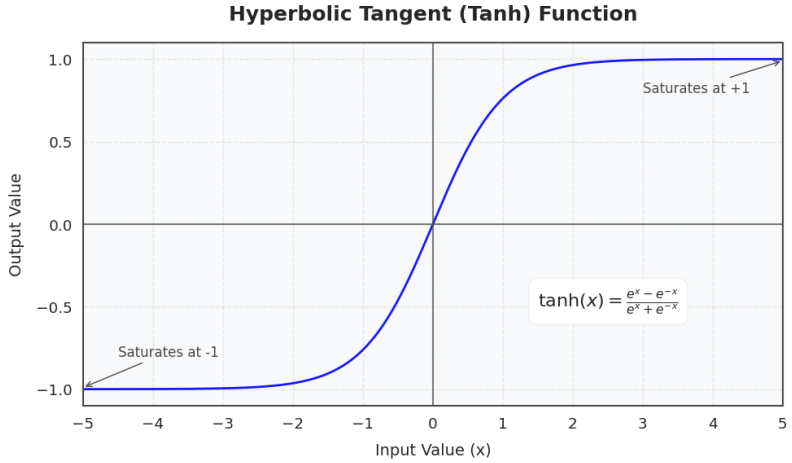
Statistics:

Neurons	Epoch	Alpha{reg}	Test.Ac	Tr.Ac	Tr.Time	Te.Time
200	200	0.9	0.8713	0.9433	0.6290	0.0141
100	200	0.9	0.8144	0.8898	0.2523	0.0044
50	200	0.9	0.8557	0.9256	0.3467	0.0070
50	200	0.5	0.8760	0.9485	0.4043	0.0008
200	200	0.5	0.8541	0.9215	0.2952	0.0071

Observations:

Παρατηρούμε ότι παίρνουμε το καλύτερο αποτέλεσμα όταν έχουμε 50 νευρώνες,200 εποχές και 0.5 ομαλοποίηση καθώς μειώνεται η διαφορά Training Ac και Test Accuracy αλλά και το Training Ac παραμένει πολύ υψηλό το οποίο θα πει ότι το Jtrain{loss} είναι πολύ χαμηλό.

Tahn Activation Function



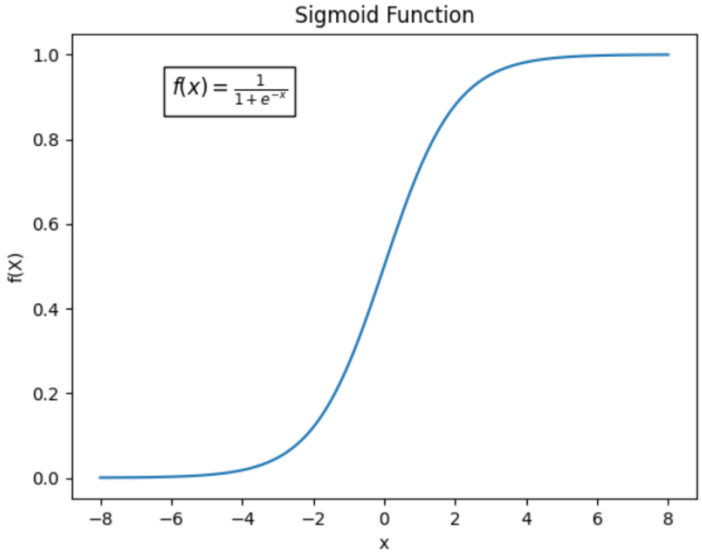
Statistics:

Neurons	Epoch	Alpha{reg}	Test.Ac	Tr.Ac	Tr.Time	Te.Time
100	200	0.9	0.8424	0.9028	0.5618	0.0028
200	200	0.9	0.8315	0.8736	0.5181	0.0130
300	200	0.9	0.8463	0.9028	3.7189	0.0122
200	100	0.9	0.8276	0.8700	0.5063	0.0110
200	300	0.9	0.8284	0.8679	0.4707	0.0060
200	100	0.5	0.8222	0.8596	0.4145	0.0053

Observations:

Την καλύτερη απόδοση σε Training Ac. Και Testing Ac την παίρνουμε στην περίπτωση των 300 νευρώνων,200 εποχών και 0.9 ομαλοποίησης όμως σε σύγκριση με όλες τις άλλες περιπτώσεις ‘έχει σχεδόν x7 χρόνο για training…Έτσι θα προτιμήσουμε το δεύτερο καλύτερο μοντέλο των 100 νευρώνων ,200 εποχών και 0.9 ομαλοποίησης. Επίσης παρατηρήθηκε ότι στους 200 νευρώνες και 0.9 ομαλοποίηση και να αυξήσουμε σε μεγάλο βαθμό το epoch ΔΕΝ υπάρχει κάποια τρομερή διαφορά.

Sigmoid Activation Function



Statistics:

Neurons	Epoch	Alpha{reg}	Test.Ac	Tr.Ac	Tr.Time	Te.Time
100	200	0.9	0.7363	0.7514	0.5625	0.0072
200	200	0.9	0.7574	0.7774	0.9078	0.0036
300	200	0.9	0.7660	0.7795	4.3715	0.0167
300	200	0.5	0.7785	0.8014	3.9265	0.0123

Observations:

Παρατήρησα ότι όταν άλλαζα των αριθμό των νευρώνων και του alpha μονάχα τότε είχα βελτιώσεις στα αποτελέσματά μου έτσι τα πειράματα με αύξηση/μείωση του epoch ήταν ασήμαντα. Τα καλύτερα αποτελέσματα τα παίρνουμε στην περίπτωση των 300 νευρώνων και της ομαλοποίησης 0.5

Bonus:Έγιναν και άλλα αντίστοιχα πειράματα τόσο με άλλους solvers όσο και με εισαγωγή δεύτερου hidden layer. Αυξομειώθηκαν παράμετροι ΑΛΛΑ τα αποτελέσματα ΕΙΤΕ έριχναν το Training Accuracy και Testing accuracy γύρο στο 70%-77% είτε το Training Accuracy έφτανε το 100% ενώ το Testing accuracy το 80% κάτι ΜΗ θεμιτό καθώς το overfitting «εκτοξεύεται» με αποτέλεσμα το μοντέλο να μαθαίνει τέλεια τα χαρακτηριστικά του training αλλά αποτυγχάνει στα άγνωστα δεδομένα!(Το οποίο είναι μεγάλο πρόβλημα)

Σχολιασμοί για τις αποδώσεις:

Όπως σχολιάστηκε και παραπάνω από τους 3εις αλγόριθμους της ενδιάμεσης εργασίας ο KNN=1 μας προσφέρει το καλύτερο test accuracy λόγω της καλής αναγνώρισης μοτίβων που έχει και του memorization.Βέβαια αυτοί οι αλγόριθμοι ΔΕΝ χρειάζονται training όπως το μοντέλο μας για αυτό έχουμε και μηδενικό training time.Στο testing ο Centroid είναι ο πιο γρήγορος από τους 3εις μονάχα για να ηττηθεί από το MLP μας.Το MLP μας έχει το καλύτερο Test accuracy αλλά και το μικρότερο Test time.Γενικά το MLP δεν βασίζεται στις αποστάσεις των παραδειγμάτων όπως ο KNN&Centroid αλλά αναλύει μια γενική δομή των εικόνων που γίνεται train.Έτσι το Test accuracy είναι πιο υψηλό γιατί αντί να «θυμάται» εικόνες όπως οι αλγόριθμοι το μοντέλο «μαθαίνει» μέσω γενικοποίησης των pattern.Επίσης γνωρίζουμε ότι ο KNN είναι ένας πολύ ευαίσθητος στο nose καθώς ένα λάθος label ή ένα λίγο αλλοιωμένο παράδειγμα μπορεί να φέρει καταστροφή κάτι που ΔΕΝ συμβαίνει στο MLP καθώς μπορεί να υπολογίσει τον μέσο όρο του θορύβου μέσω της κλίσης και μάθησης από αυτην.

Bonus:

Αυξάνοντας το μέγεθος του αρχικού,σχετικά μικρού μου dataset παρατηρώ ότι το φαινόμενο του overfitting σε πολλαπλά πειράματα χαμηλώνεται καθώς το training accuracy και το testing accuracy έχουν λιγότερη διαφορά μεταξύ τους.

References Used:

1.for resize/flattening/Pca:
https://eitca.org/artificial-intelligence/eitc-ai-dlpp-deep-learning-with-python-and-pytorch/neural-network/building-neural-network/examination-review-building-neural-network/why-do-we-need-to-flatten-images-before-passing-them-through-the-network/

https://www.geeksforgeeks.org/machine-learning/impact-of-image-flattening/
https://youtu.be/pj9-rr1wDhM?si=3n9iGKQkBG9GFg8s
https://www.geeksforgeeks.org/machine-learning/numpy-ndarray-flatten-function-python/

https://www.geeksforgeeks.org/data-analysis/principal-component-analysis-pca/

2.For K-nn n Centroid building:
https://youtu.be/rTEtEy5o3X0?si=xy6ubcCMP4ame58z

https://youtu.be/ngLyX54e1LU?si=PtWtj2upiCHpxjMp

https://youtu.be/xtaom__-drE?si=kegEwtY9mCihAK8t

https://youtu.be/G0OoVKgHTpY?si=Ah4qvMmUgZeDp_n0

3.Commands/Libraries:
https://realpython.com/ref/stdlib/os/
https://www.geeksforgeeks.org/pandas/pandas-tutorial/

https://scikit-learn.org/stable/