ΜΕΛΗ

Όνομα	**Επώνυμο**	**ΑΜ**
Άννα	Πετρίδου	1115201600135
Μιχάλης	Ανδρουλάκης	1115201600004

Α' Μερος

Σχολιασμος Αποτελεσμάτων:

Για το Α μερος της εργασιας εγιναν πολλαπλά πειραματα και εκτελεσεις, αλλάζοντας μια υπερπαράμετρο καθε φορα και κρατωντας ολες τις αλλες σταθερες. Οι υπερπαράμετροι για τις οποιες έγιναν εκτελέσεις ειναι ο αριθμος των φιλτρων, ο αριθμος των hidden layers, το μεγεθος των φιλτρων (θεωρουμε πως ειναι τετραγωνο, οποτε σα μεγεθος λαμβανουμε εναν αριθμο που ειναι η μια πλευρα του), το πληθος των εποχων, το πληθος των batches και φυσικα το learning rate.

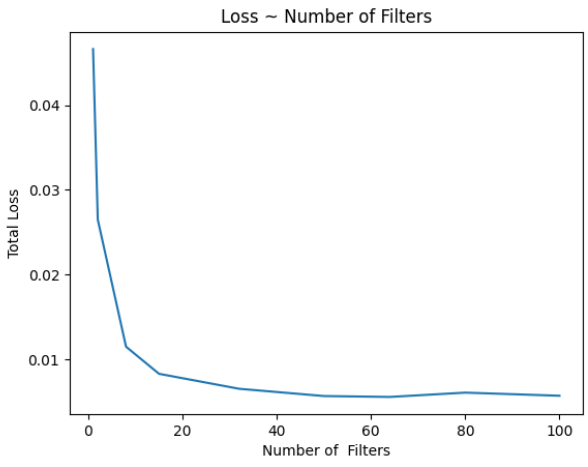
Σαν default τιμες για ολες τις εκτελεσεις επιλεξαμε τις παρακατω, επειδη εδιναν ενα αντιπροσωπευτικο αποτελεσμα σε ικανοποιητικο χρονο ωστε να μπορουν να γινουν τα πειραματα για 9 εκτελεσεις το καθενα.

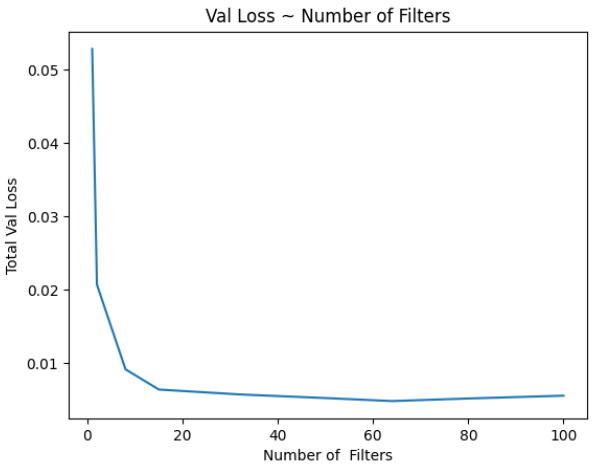
Filters:32 Hidden Layers:5 Filter Size:3 (δηλ. 3x3) Epochs:10 Batches:100 Learning Rate:0.001

Χρησιμοποιωντας αυτες τις default τιμες, καναμε 9 εκτελεσεις για καθε υπερπαραμετρο, στις οποιες κραταγαμε ολες τις παραπανω τιμες σταθερες, εκτος απο αυτη της εκαστοτε υπερπαραμετρου που εξεταζαμε.

Παρακατω φαινονται και γραφικα τα αποτελεσματα, στον αξονα Χ οι τιμες της εκαστοτε υπερπαραμετρου και στον Υ οι τιμες του loss ή του val_loss:

Για τον αριθμό των φιλτρων σε σχεση με το loss στο training και το loss στο validation:

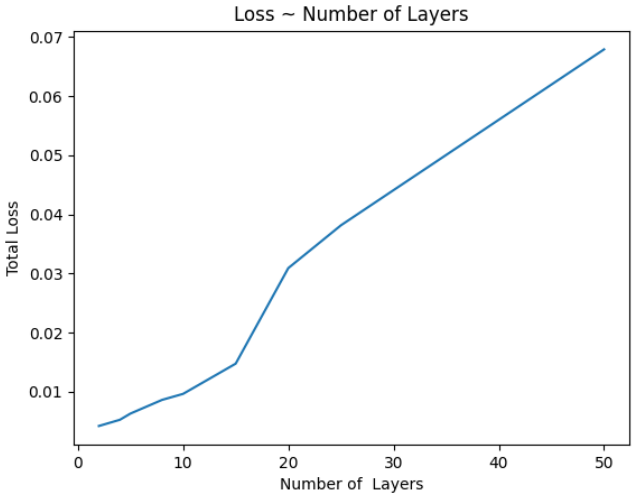


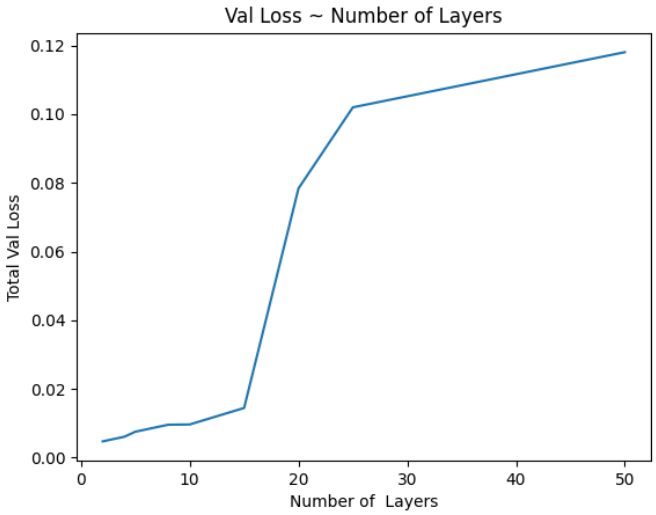


Παρατηρουμε οτι τοσο το loss οσο και το val loss μειωνονται οσο αυξανεται η τιμη των φιλτρων, ενω παρατηρουμε οτι σταθεροποιειται μετα απο καποια στιγμη και

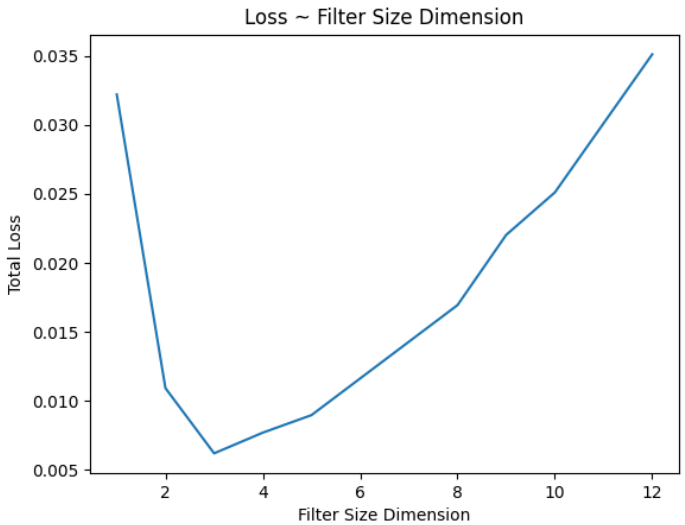
ενδεχομενως αυξανεται και λιγο. Η ιδανικη τιμη για το πληθος των φιλτρων παρατηρουμε πως ειναι γυρω στο 60-80.

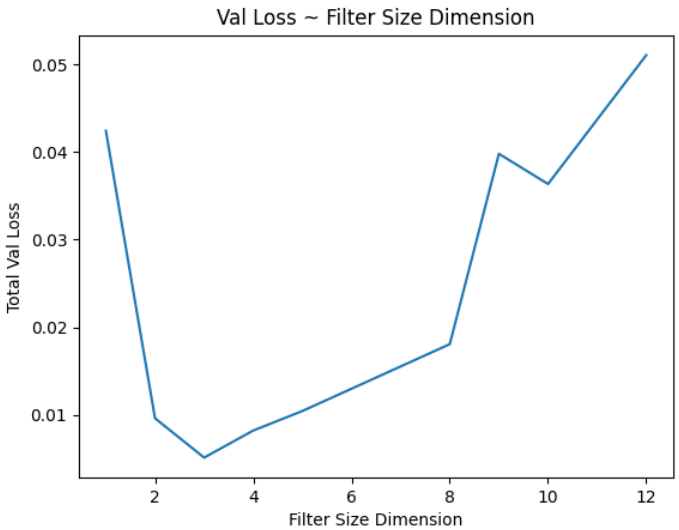
Για τις τιμες των hidden layers, παρατηρουμε οτι η τιμη του loss αυξανεται σχετικα σταθερα οσο αυξανονται τα επιπεδα, καθως η τιμη του val loss αυξανεται πιο αποτομα. Και τα δυο φαινεται να αρχιζουν τη μεγαλη αυξηση μετα τα 15 περιπου επιπεδα και οπως παρατηρουμε ειναι αυξουσες οι καμπυλες και συνεπως οσο λιγοτερα τα επιπεδα, τοσο λιγοτερα και τα loss και val loss. Ιδανικη τιμη θα ηταν καποια μικροτερη του 15.



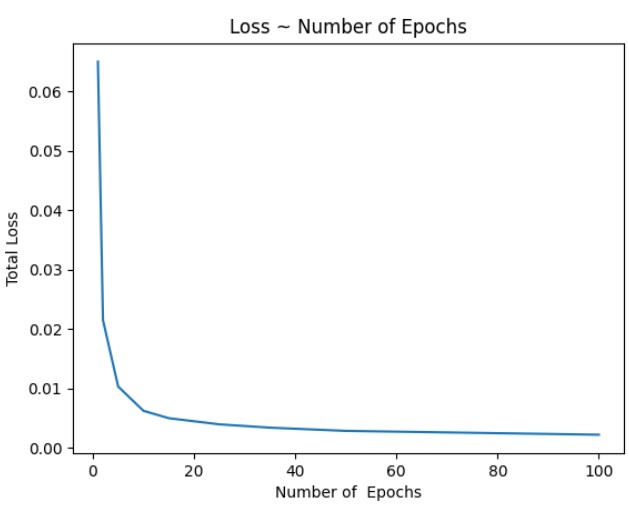


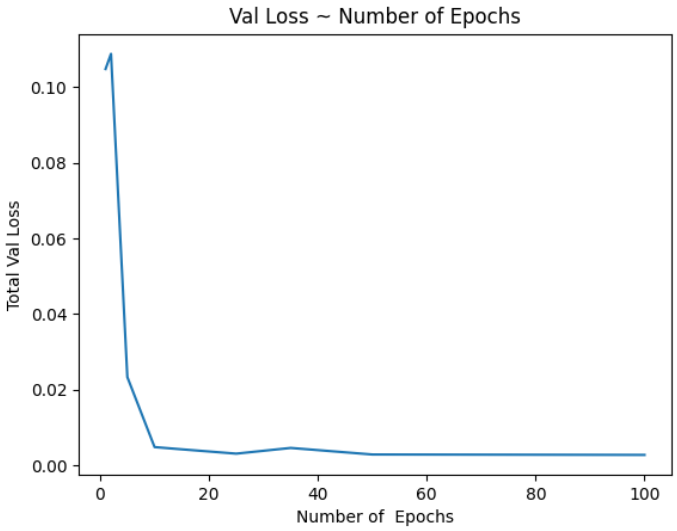
Για τις τιμες του filter size, παρατηρουμε οτι το loss και το val loss μειωνονται οσο αυξανεται το μεγεθος μεχρι περιπου το 3, που ειναι το τοπικο ελαχιστο. Επειτα βλεπουμε πως αυξανονται οσο αυξανεται και το filter size. Το val loss βλεπουμε πως ακολουθει την ιδια πορεια με το loss εκτος απο οταν ειναι στο 9 το μεγεθος, οπου τοτε μειωνεται λιγο και ξαναυξανεται. Ιδανικη τιμη παρατηρουμε πως ειναι το 3.



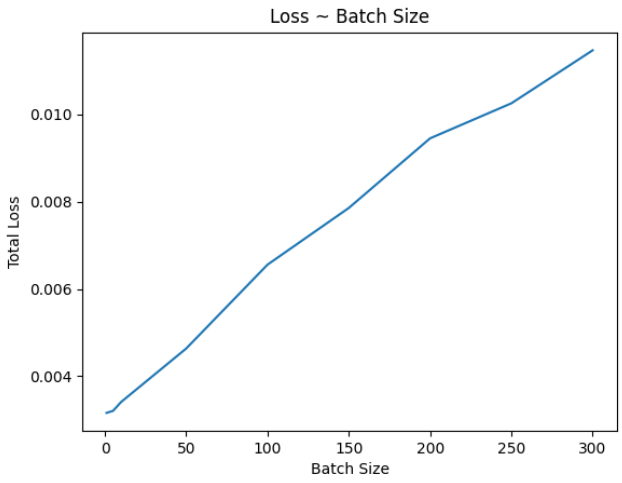


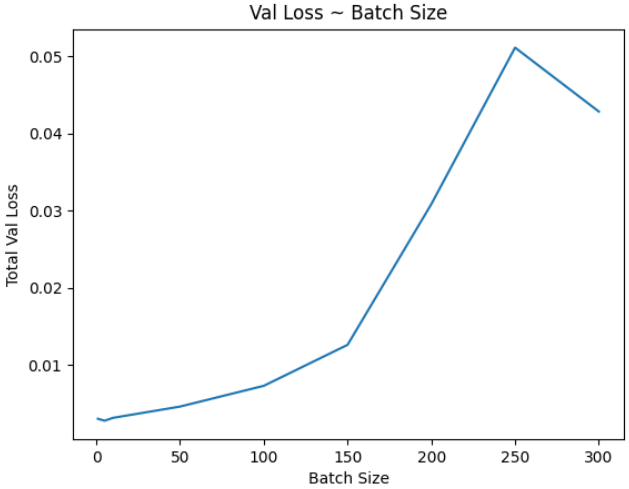
Για το πληθος των εποχων, παρατηρουμε οτι το loss και το val loss μειωνονται αποτομα οσο αυξανεται το πληθος των εποχων μεχρι τις 20 εποχες, οπου απο εκει και επειτα μειωνεται το loss και το val loss με πολυ πιο αργο ρυθμο. Φθινει, συνεπως το loss και το val loss μειωνονται οσο περισσοτερες ειναι οι εποχες, παρολα αυτα το val loss παρατηρουμε πως πιανει μια αρκετα χαμηλη τιμη στις 10 εποχες και φυσικα μειωνεται και ακομα περισσοτερο μετα τις 60.



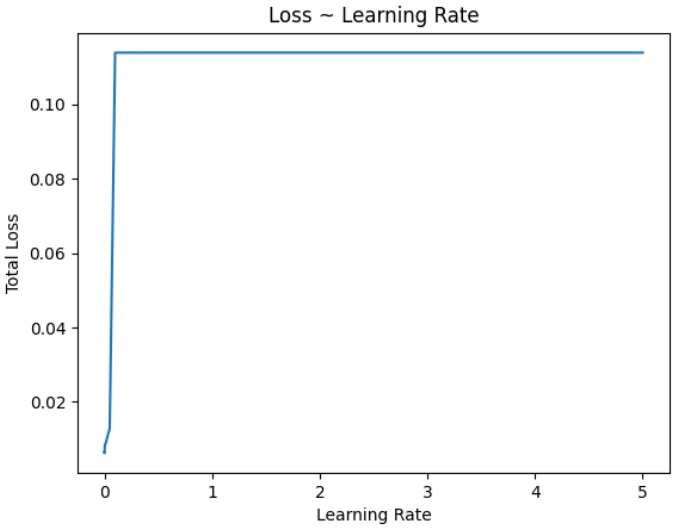


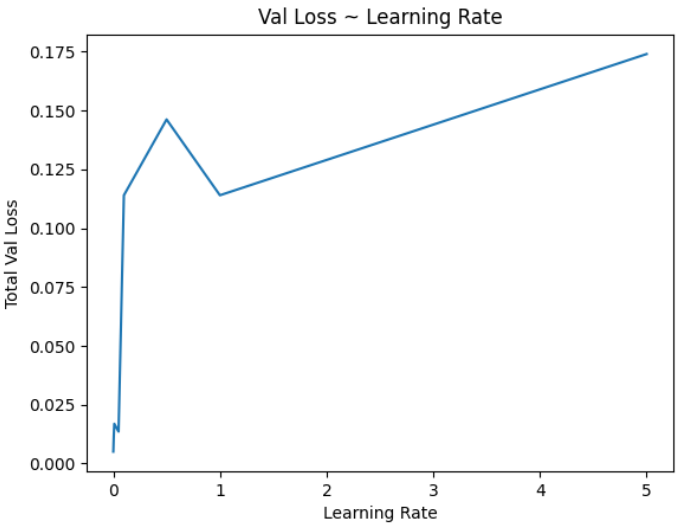
Για τον αριθμο των batches, παρατηρουμε οτι το loss αυξανεται οσο αυξανεται και ο αριθμος των batches. Το val loss αυξανεται με πιο αργους ρυθμους μεχρι τα 150 batches, μετα εχουμε αποτομη αυξηση και μετα τα 250 batches, το val loss αρχιζει παλι να μειωνεται. Η ιδανικη τιμη ειναι μια πολυ χαμηλη τιμη, οσο χαμηλοτερη, τοσο το καλυτερο, ιδανικα μικροτερη των 150.





Για το learning rate παρατηρουμε πως το loss παραμενει σταθερο μετα το 0,0001-0,001. Αντιθετα το val loss αυξανεται μεχρι περιπου το 0,5, επειτα μειωνεται και ξαναυξανεται μετα το 1. Ιδανικη τιμη παρατηρουμε πως ειναι το 0,0001-0,001.





Κατάλογος:

autoencoder.py , graphs.py , main.py , reading.py , user_input.py

Σχολιασμος Κωδικα:

Στο αρχειο graphs.py, υλοποιειται η εκτυπωση των γραφικων παραστασεων, δυο γραφικες παραστασεις για καθε υπερπαραμετρο, μια για loss-υπερπαραμετρο και μια για val loss-υπερπαραμετρο.
Στο αρχειο reading.py υλοποιειται μια readMNIST.
Στο αρχειο user_input.py υλοποιειται μια συναρτηση ωστε να ελεγχεται πως το input που λαμβανεται απο το χρηστη ειναι ειτε εγκυρος(θετικος) ακεραιος ή δεκαδικός, αλλιώς να ξαναζητάει απο το χρηστη νεο input. Αυτη η συναρτηση χρησιμοποιειται στη main, στο διαβασμα των υπερπαραμετρων που δινει ο χρηστης.
Στο αρχειο autoencoder.py , εχουμε τον encoder και decoder, οπως αυτα περιγραφηκαν στις διαλεξεις. Στον encoder εχουμε χωρισει τα layers δια τρια και εχουμε το 1/3 πριν το πρωτο maxpooling, το 1/3 πριν το δευτερο maxpooling και το 1/3 (ή οσο περισσευει αν δε διαιρειται ακριβως) στο τελος. Χρησιμοποιησαμε το dropout για να αποφευχθει το overfitting οπως συζητηθηκε στο μαθημα. Μετα απο πειραματα, καταληξαμε οτι η καλυτερη τιμη για το dropout μας ειναι το 0.2. Στην περιπτωση που τα layers ειναι λιγοτερα απο 3,(δηλαδη 2) τοτε τα σπαμε σε ενα πριν καθε maxpooling.

Η ιδια λογικη ακολουθηθηκε και για τον decoder, βασιζομενοι παντα στις σημειωσεις του φροντιστηριου.
Στο αρχειο main.py ξεκιναμε με το ορισμα που δεχομαστε, δηλαδη το dataset, το οποιο παιρνουμε και χρησιμοποιωντας τη readMNIST μας απο το reading.py, το αποθηκευουμε σε ενα images_collection. Επειτα κανουμμε split το dataset σε training set και evaluation set, χρησιμοποιωντας την “train_test_split” . Δημουργουμε πινακες για καθε υπερπαραμετρο και το συνολικο loss και το συνολικο validation loss, ωστε να κραταμε τις τιμες για να φτιαξουμε τις γραφικες μετα απο πολλαπλες εκτελεσεις του προγραμματος. Στη συνεχεια δινονται απο το χρηστη οι υπερπαραμετροι, οι οποιες ελεγχονται με το user_input.py μας και επειτα δημιουργειται, εκπαιδευεται και ελεγχεται το μοντελο μας, χρησιμοποιωντας τις σημειωσεις του φροντιστηριου. Οταν τελειωνει ολη η διαδικασια,δινονται στο χρηστη 3 επιλογες για να επιλεξει πως θα προχωρησει. Μπορει 1) να ξανατρεξει το προγραμμα με αλλες υπερπαραμετρους, 2) να εμφανισει τις γραφικες και να τερματισει το προγραμμα ή 3)να αποθηκευσει το μοντελο και να τερματισει το πτογραμμα.

Για τη readMNIST και το normalization, χρησιμοποιησαμε αυτες τις πηγες:
https://medium.com/the-owl/converting-mnist-data-in-idx-format-to-python-numpy-array-5cb9126f99f1
https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-dataset-54c35b77a38d

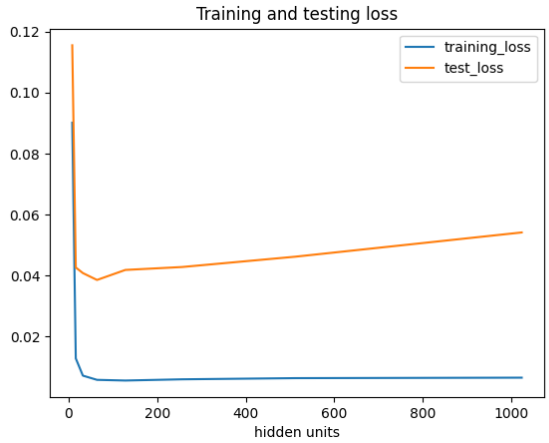
Οδηγίες Χρήσης:
Εντολή εκτέλεσης ενδεικτικά:
$ python3.8 autoencoder.py -d t10k-images.idx3-ubyte

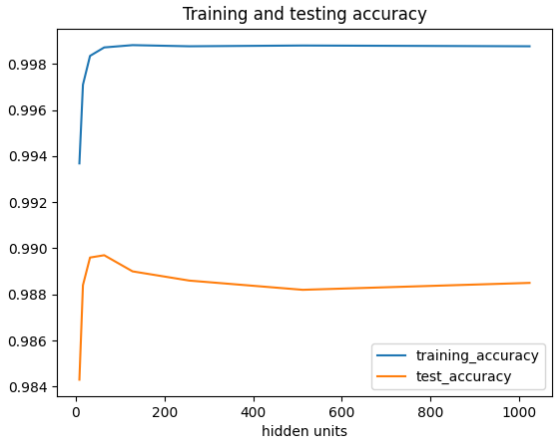
Μόλις ξεκινάει το πρόγραμμα, δέχεται τις τιμες των υπερπαραμέτρων (filters,layers,filter size, epochs, batches, learning rate) και όταν τελειωσει το προγραμμα βγαινουν οι 3 επιλογες όπως αυτες περιγραφονται στην εκφωνηση. Ο χρηστης μπορει να πατησει 1, αν επιθυμει να εκτελεσει ξανα το προγραμμα με αλλες παραμετρους, 2 αν επιθυμει να εμφανιστουν οι γραφικες παραστασεις και να τελειωσει το προγραμμα ή 3 να αποθηκευτει το μοντέλο και να τελειωσει το προγραμμα.

Β’ ΜΕΡΟΣ

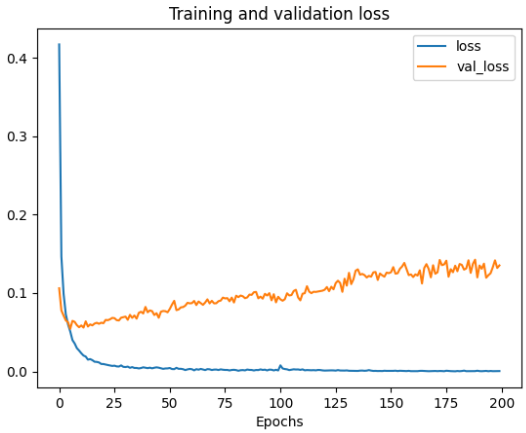
Σχολιασμός: Για το Β μερος τα καλυτερα αποτελεσματα τα πηραμε για FC Units=64, Epochs=20 , Batch Size=64 , Learning Rate=0.00005 καθως τοτε εχουμε

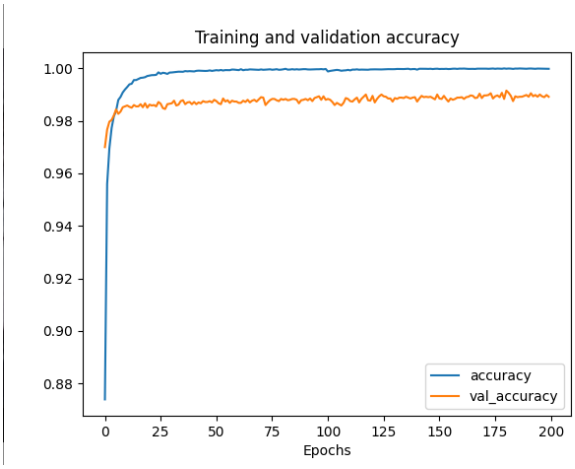
το ελαχιστο loss και validation loss και τα μεγιστα accuracy και validation accuracy.Το διαγραμμα των units σε συναρτηση με τα training-test loss και training-test accuracy βγαζει τις βελτιστες τιμες στη τιμη fc units=64.Αυτα τα διαγραμματα εγιναν με epochs=20 , Batch Size=64 και Learning Rate=0.00005.



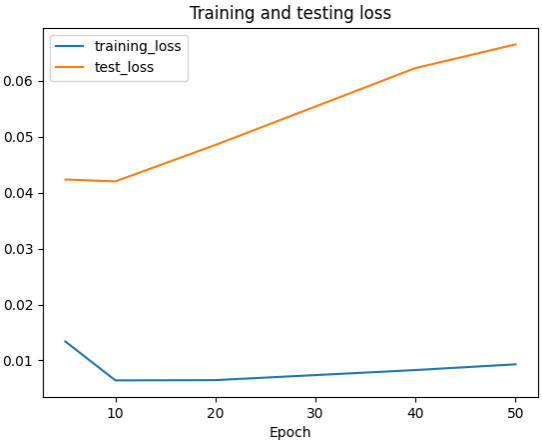


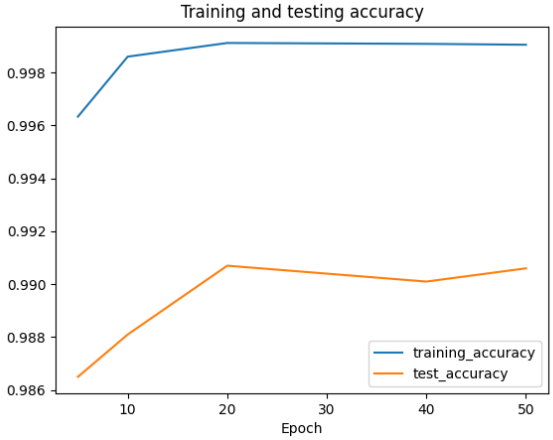
Ακόμα,το μοντελο μας συγκλινει αρκετα γρηγορα οποτε δε μας ηταν απαραιτητος ο αριθμος Epochs=100.Εχουμε βεβαια μερικες ενδεικτικες γραφικες παραστασεις για αυτο τον αριθμο Epochs οπως θα δειτε παρακατω:





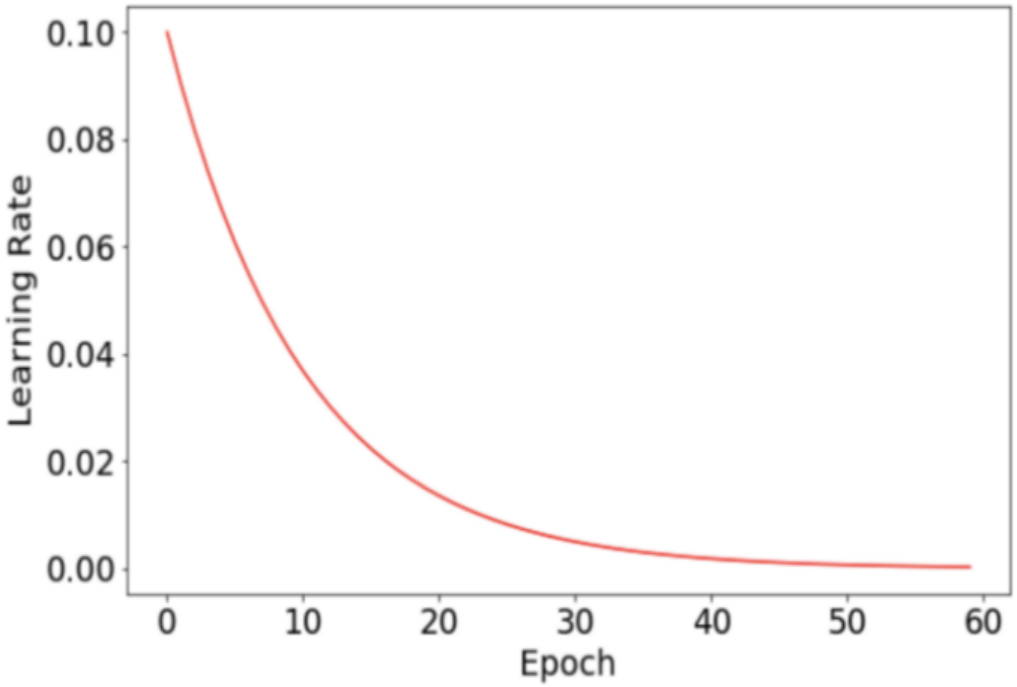
Βρηκαμε ετσι τον βελτιστο αριθμο Epochs , στον αριθμο 20,δηλαδη 20 εποχες στο πρωτο σταδιο εκπαιδευσης και αλλες 20 στο δευτερο πετυχαινοντας βελτιστα αποτελεσματα ακομα και καλυτερα απο τα ενδεικτικα.Η ελαχιστοποιηση λοιπον των training-test loss και η βελτιστοποιηση των training-test accuracy γινεται στη τιμη 20,οπως φαινεται στα παρακατω διαγραμματα που εγινε με Fc units=64 , Batch Size=64 και Learning Rate=0.00005.



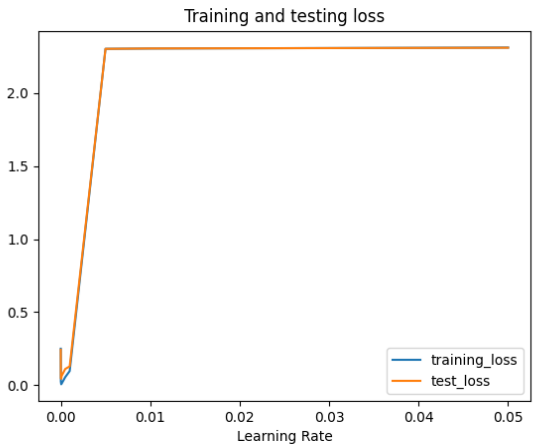


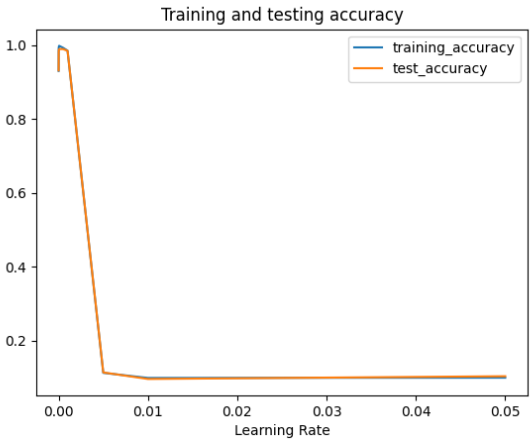
Ακόμη με ερευνα στο διαδικτυο αλλα και με δικα μας πειραματα καταληξαμε οτι η βελτιστη τιμη για το Learning Rate ειναι το 0.00005 καθως τοτε εχουμε , οπως βλεπετε παρακατω , ελαχιστα loss και μεγιστα accuracy.Τα πειραματα και εδω γινανε με Fc units=64 , Epochs=20 και Batch Size=64.

Εδω λοιπον ειναι ενα διαγραμμα του Learning Rate συναρτησει των εποχων που βρηκαμε στο διαδικτυο:

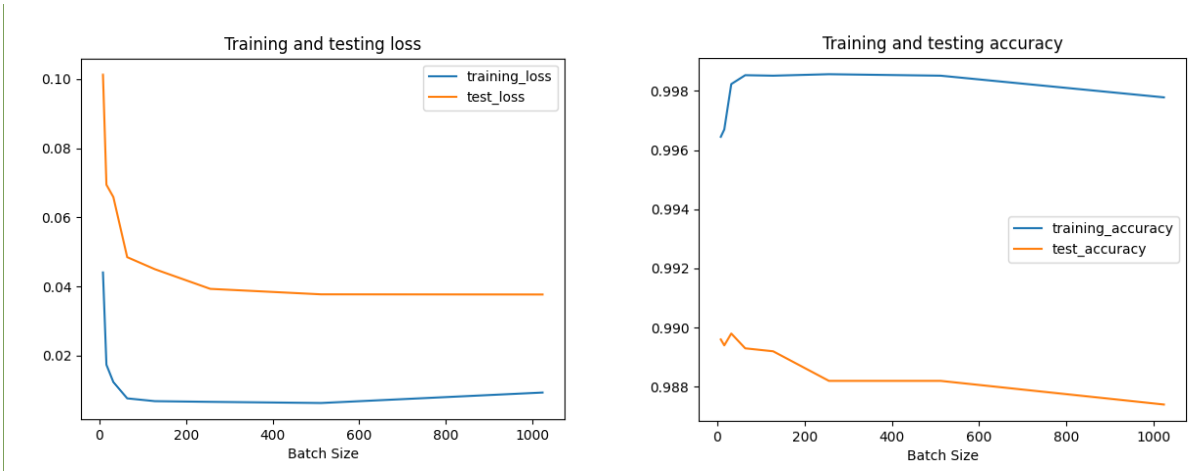


Και εδω ειναι τα διαγραμματα με τα δικα μας πειραματα

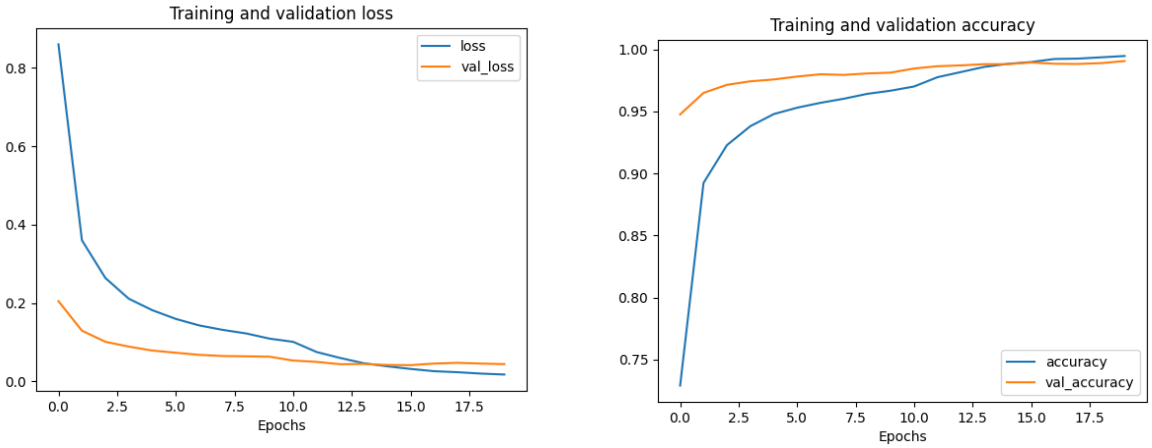




Τελος,για το batch size υστερα απο πειραματα,τις καλυτερες τιμες , για train-test accuracy και train-test loss σε συνδυασμο με τα καλυτερα classification reports , τις παιρνουμε για batch size=64.
Τα πειραματα και εδω εγιναν με Fc units=64 , Epochs=20,Learning Rate=0.00005.



Εδω εχουμε μερικες γραφικες παραστασεις για (validation) loss και (validation) accuracy με τιμες: fc units=64 , Epochs=10 , Batch Size=64 και Learning Rate=0.00005 :



Χρησιμοποιησαμε τον RMSprop optimizer και οχι τον Adam καθως με διαφορα τεστ που υλοποιησαμε παιρναμε καλυτερα αποτελεσματα με αυτον.

Παρακατω ειναι διαφορα classification reports για διαφορα Fc units.
Αυτα τα reports προεκυψαν με Epochs=20 , Batch Size=64 και Learning Rate=0.00005:

For the experiment with 32 Hidden Units

Test loss: 0.04085105285048485
Test accuracy: 0.9896000027656555
Found 9896 correct labels
Found 104 incorrect labels

	precision	recall	f1-score	support
Class 0	0.99	0.99	0.99	980
Class 1	0.99	0.99	0.99	1135
Class 2	0.99	0.99	0.99	1032
Class 3	0.99	0.99	0.99	1010
Class 4	0.99	0.99	0.99	982
Class 5	0.99	0.99	0.99	892
Class 6	0.99	0.99	0.99	958
Class 7	0.99	0.99	0.99	1028
Class 8	0.99	0.99	0.99	974
Class 9	0.99	0.98	0.98	1009
micro avg	0.99	0.99	0.99	10000
macro avg	0.99	0.99	0.99	10000
weighted avg	0.99	0.99	0.99	10000
samples avg	0.99	0.99	0.99	10000

For the experiment with 64 Hidden Units

Test loss: 0.04849178344011307
Test accuracy: 0.989300012588501
Found 9893 correct labels
Found 107 incorrect labels

	precision	recall	f1-score	support
Class 0	0.99	1.00	0.99	980
Class 1	1.00	0.99	0.99	1135
Class 2	0.99	0.99	0.99	1032
Class 3	0.99	0.99	0.99	1010
Class 4	1.00	0.98	0.99	982
Class 5	0.99	0.99	0.99	892
Class 6	0.99	0.99	0.99	958
Class 7	0.99	0.99	0.99	1028
Class 8	0.99	0.99	0.99	974
Class 9	0.99	0.99	0.98	1009
micro avg	0.99	0.99	0.99	10000
macro avg	0.99	0.99	0.99	10000
weighted avg	0.99	0.99	0.99	10000
samples avg	0.99	0.99	0.99	10000

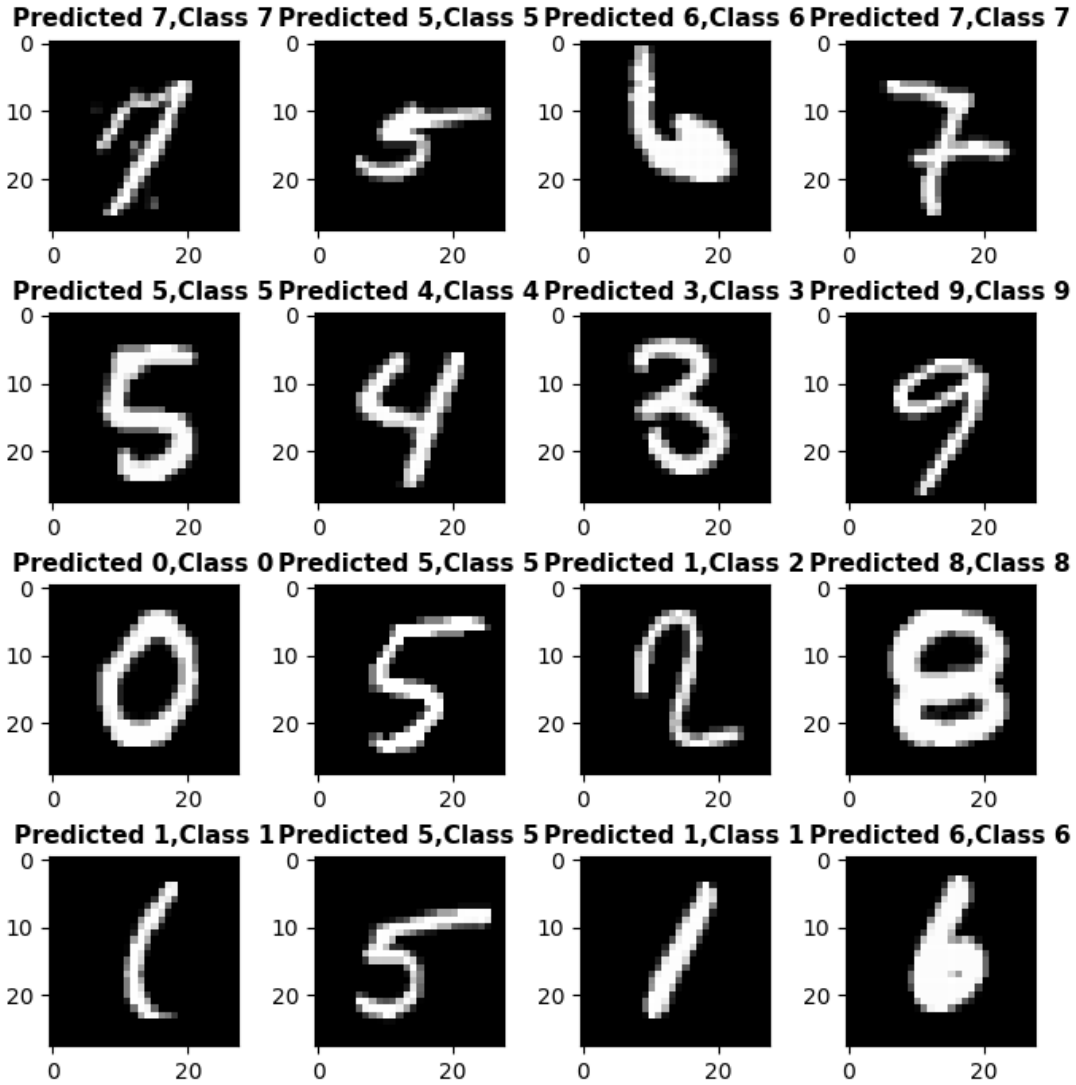
For the experiment with 128 Hidden Units

Test loss: 0.041875168681144714
Test accuracy: 0.9890000224113464
Found 9890 correct labels
Found 110 incorrect labels

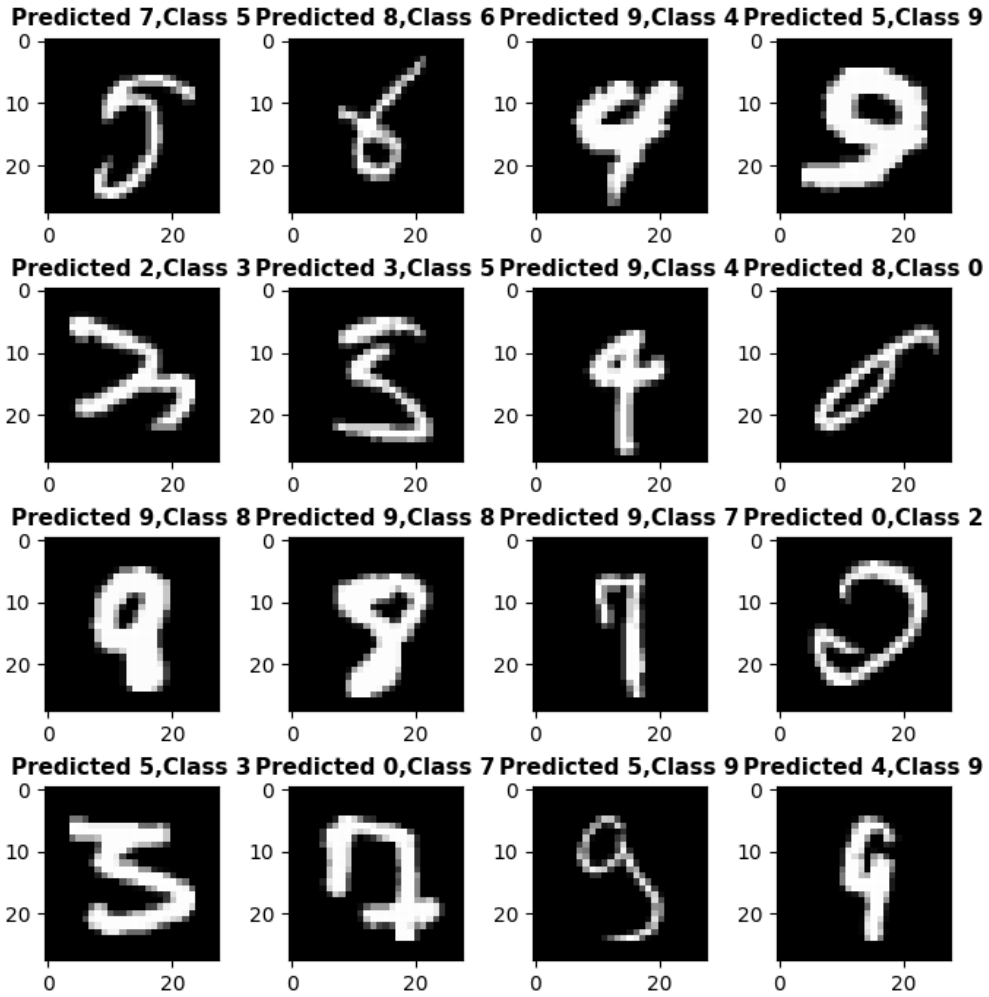
	precision	recall	f1-score	support
Class 0	0.99	0.99	0.99	980
Class 1	1.00	1.00	1.00	1135
Class 2	0.99	0.99	0.99	1032
Class 3	0.99	0.99	0.99	1010
Class 4	1.00	0.99	0.99	982
Class 5	0.98	0.99	0.99	892
Class 6	0.99	0.99	0.99	958
Class 7	0.99	0.99	0.99	1028
Class 8	0.99	0.98	0.99	974
Class 9	0.99	0.98	0.98	1009
micro avg	0.99	0.99	0.99	10000
macro avg	0.99	0.99	0.99	10000
weighted avg	0.99	0.99	0.99	10000
samples avg	0.99	0.99	0.99	10000

Ακομα εχουμε και μερικα classification εικονων.
Τα παρακατω εγιναν με Fc units=64 , Epochs=20 , Batch Size=64 και Learning Rate=0.00005.

Το πρωτο classification αφορα σωστη προβλεψη εικονων



Το δευτερο classification αφορα λαθος προβλεψη εικονων



Οδηγίες Χρήσης: Για το Β μερος της εργασιας ο χρηστης πρωτα πρεπει να δωσει 5 ορισματα στη γραμμη εντολων,επειτα,θα αρχισει την εκπαιδευση των πειραματων του.Το προγραμμα θα του ζητησει: τον αριθμο των hidden units του fully connected layer (integer) , τον αριθμο των epochs των 2 φασεων (integer) , το batch size των δυο φασεων (integer) , και το learning rate των δυο φασεων (float).
Επειτα το προγραμμα κανει train και οταν τελειωσει δινει στο χρηστη 4 επιλογες:Ειτε να τρεξει ενα νεο πειραμα με νεα τετραδα υπερπαραμετρων , ειτε να εμφανισει τα classification reports (recall,f1score,precision κλπ) ολων των πειραματων μεχρι

στιγμης καθως και διαγραμματα των accuracy και loss σε συναρτηση με τις υπερπαραμετρους , ειτε να κατηγοριοποιησει τις εικονες με ενα πειραμα της επιλογης του , ειτε να τερματισει το προγραμμα.
Στη πρωτη περιπτωση ο χρηστης πληκτρολογει ξανα hidden units , epochs , batch size , learning rate.
Στη δευτερη περιπτωση μετα την εμφανιση των classification reports ο χρηστης εχει τη δυνατοτητα να επιλεξει να εμφανισει τεσσαρων τυπων διαγραμματα.Αυτα τα διαγραμματα εχουν στον αξονα y τα train,test loss και τα train,test accuracy και στον αξονα x μια απο τις τεσσερις υπερπαραμετρους.Εκει ο χρηστης πρεπει να επιλεξει 1,2,3 ή 4 αναλογως ως προς ποια υπερπαραμετρο θελει να εμφανισει τα διαγραμμα των loss και accuracy.Μετα την επιλογη εμφανιζονται στην οθονη 1 διαγραμμα με τα train-test loss ως προς την επιλεγμενη υπερπαραμετρο και ενα διαγραμμα train-test accuracy ως προς την ιδια επιλεγμενη υπερπαραμετρο.Μετα το προγραμμα δινει στο χρηστη 2 επιλογες,αν θελει να εμφανισει και αλλα διαγραμματα ως προς υπερπαραμετρους ή αν θελει να συνεχισει.Στη περιπτωση που πατησει 1 αντιστοιχα παλι εχει 4 επιλογες διαγραμματων.Στη περιπτωση που πατησει 2 το προγραμμα του δινει παλι τις 4 αρχικες επιλογες.
Στη τριτη περιπτωση(image classification) ο χρηστης πρεπει να πληκτρολογησει 4αδα υπερπαραμετρων ενος υπαρχοντος πειραματος ωστε να γινει το classify με αυτο το πειραμα.Επειτα ρωταμε τον χρηστη αν θελει να απεικονηθει ενα δειγμα 16 σωστων ή λανθασμενων εικονων επιλεγοντας 1 ή 2 αντιστοιχα.Επειτα απο αυτο ο χρηστης εχει παλι τις 4 αρχικες επιλογες.
Για τις δυο φασεις του train χρησιμοποιουνται τα ιδια fc units,epochs,batch size,learning rate και δε ζηταμε απο τον χρηστη να τα πληκτρολογησει ξανα στη 2η φαση διοτι διαπιστωσαμε οτι δεν εχουν νοημα οι διαφορετικες παραμετροι στις 2 φασεις και επισης αμα σε καθε φαση ζητουσαμε απο τον χρηστη νεες υπερπαραμετρους τοτε αυτο θα ηταν δυσχρηστο στις επιλογες εμφανισης Image Classification.Φανταστειτε για να εμφανισει τις εικονες ενος πειραματος ο χρηστης να πρεπει να πληκτρολογησει 8 τιμες (4 της πρωτης φασης και 4 της δευτερης).Για αυτο το λογο το παραλειψαμε.

Καταλογος:

classification/reading_and_converting.py :
Αυτο το αρχειο περιεχει τις συναρτησεις readingArguements (για το διαβασμα των αρχειων της γραμμης) και το file_normalization (το οποιο κανονικοποιει σε πινακες τα αρχεια training και test οπως πχ να φιλτραρει τα pixel των set αρχειων στο 0-1 διαιροντας τα με 255 και να φτιαξει τα αρχεια labels να ειναι της μορφης 28*28*1).
classification/classifier_building.py :
Αυτο το αρχειο περιεχει τη συναρτηση build_model η οποια χρησιμοποιεται για τη κατασκευη του classifier(δηλαδη κρατωντας τον encoder απτον autoencoder και προσθετοντας ενα στρωμα flatten,ενα fully connected layer,ενα dropout και ενα fully connected layer 10 στρωματων).
classification/classification_results :
Αυτο το αρχειο περιεχει τις συναρτησεις print_classification_reports (η οποια ειναι για να εκτυπωνονται ολα τα classification reports ολων των πειραματων που εχει τρεξει μεχρι τωρα ο χρηστης) , display_graph_option (η οποια παρουσιαζει τις

επιλογες διαγραμματων που μπορει να εκτυπωσει ο χρηστης) , printGraphs (που εκτυπωνει τις γραφικες παραστασεις ως προς την επιλεγμενη υπερπαραμετρο) , image_classification (που επιλεγει ο χρηστης ενα πειραμα ωστε να εκτυπωσει ενα δειγμα εικονων και το που ενταχθηκαν).
classification/classification.py :
Οπου εδω καλουνται ολες οι προαναφερθεισες συναρτησεις κανοντας επισης compile,fit,evaluate το μοντελο με τις υπερπαραμετρους που επελεξε ο χρηστης.

Εκτέλεση: python3 classification.py –d train-images.idx3-ubyte –dl train-labels.idx1-ubyte –t t10k-images.idx3-ubyte –tl t10k-labels.idx1-ubyte –model my_h5_model.h5

Github link : https://github.com/Omada53/project2