Νευρωνικά Δίκτυα - Βαθιά Μάθηση

Τρίτη Υποχρεωτική Εργασία

Ευαγγελία Δημαρίδου ΑΕΜ: 10410

I. Εισαγωγή

Το θέμα της παρούσας εργασίας είναι να υλοποιήσουμε ένα δίκτυο αυτοκωδικοποίησης (Autoencoder), το οποίο θα επιλύει το πρόβλημα ανακατασκευής ψηφίων του συνόλου δεδομένων της MNIST. Ακόμη το δίκτυο εκπαιδεύτηκε και για να κατασκευάζει το επόμενο ψηφίο καθώς και να αθροίζει δύο ψηφία. Για την εκπαίδευση του δικτύου χρησμοποιήθηκαν διάφορες προσεγγίσεις οι οποίες θα παρουσιαστούν αναλυτικά.

II. Autoencoder

Προκειμένου να δημιουργηθεί ο Autoencoder, χρησιμοποιήθηκε ο πολυστρωματικός perceptron της πρώτης εργασίας. Συγκεκριμένα χρησιμοποιήθηκε η from scratch υλοποίηση του. Προκειμένου το νέο δίκτυο να είναι δίκτυο αυτοκωδικοποίησης, πραγματοποιήσαμε διάφορες αλλαγές.

- Τα layers του δικτύου χωρίστηκαν σε encoder και decoder, ενώ το πρώτο layer είναι ξεχωριστό με την ονομασία first_layer.

- Υπάρχει μία μεταβλητή bottle_neck η οποία δείχνει στο latent space προκειμένου να γνωρίζουμε που δεν θα εφαρμοστεί η συνάρτηση ενεργοποίησης relu.

- Οι συναρτήσεις feed_forward() και back_propagation() προσαρμόστηκαν καθώς πλέον τα hidden layers είναι χωρισμένα σε δύο μεταβλητές αλλά υπάρχει προφανώς επικοινωνία μεταξύ τους.

- Στο layer εξόδου εφαρμόστηκε η συνάρτηση ενεργοποίησης sigmoid. Οι τιμές που παίρνει αυτή η συνάρτηση είναι μεταξύ 0 και 1, και τα δεδομένα μας έχουν επίσης τιμές απο το 0 μέχρι το 1. Έτσι υποχρεωτικά και η έξοδος πρέπει να συμβαδίζει με την είσοδο αφού προσπαθούμε να ανακατασκευάσουμε το δείγμα εισόδου.

- Στην εργασία κατηγοριοποίησης χρησιμοποιούσαμε την συνάρτηση softmax. Έτσι, το delta του output layer ήταν ίσο με το error του output δηλαδή την διαφορά μεταξύ του target και του output. Τώρα όμως που χρησιμοποιούμε την sigmoid, εφαρμόζουμε τον τύπο:

$$\delta_j = e(j) * \varphi'_j(u_j)$$

- Θεωρούμε ακόμη ότι ο χρήστης δίνει τα σωστά και συμμετρικά layers ενώ τα input και output size είναι ίσα, και μάλιστα 784 στην περίπτωση της MNIST.

- Χρησιμοποιούμε mean squared error (MSE) και έτσι τροποποιούνται οι εξισώσεις του loss.

$$loss = \frac{\sum_{\kappa=1}^{size}(target(k) - output(k))^2}{2 * size}$$

- Για validation set, σε αυτή την εργασία χρησιμοποιήσαμε το test set. Επειδή είχαμε πραγματοποιήσει τα πειράματα με αυτό το σύνολο δεδομένων στην αρχή δεν το αλλάξαμε. Κανονικά τα δύο σύνολα πρέπει να είναι διαφορετικά, όπως και στην πρώτη εργασία, αλλά εφόσον δεν τα χρησιμοποιήσαμε για να βρούμε το καλύτερο μοντέλο, αλλά για μία εικόνα της προόδου του, το κρατήσαμε έτσι.

III. Ανακατασκευή Ψηφίου

Ο Autencoder λοιπόν που δημιουργήσαμε, χρησιμοποιήθηκε για την ανακατασκευή των ψηφίων της MNIST. Ο κώδικας αυτός βρίσκεται στο αρχείο autoencoder_reconstruct.py

A. Φόρτωση και Επεξεργασία Δεδομένων

Προκειμένου να φορτώσουμε τα δεδομένα μας, δημιουργούμε μία συνάρτηση load_mnist(). Σε αυτή την εργασία, επιλέγουμε να χρησιμοποιήσουμε την βιβλιοθήκη keras. Αφού προσαρμόσουμε τα μεγέθη των δεδομένων, διαιρούμε τα δείγματα εκπαίδευσης και ελέγχου με το 255 για λόγους numerical stability. Σε αντίθεση με τον κατηγοριοποιητή, εδώ δεν χρειαζόμαστε την κλάση του κάθε δείγματος, αφού το ίδιο το νευρωνικό δεν θα αναγνωρίσει κάποιο ψηφίο, απλά θα μάθει να το αναπαράγει. Επομένως η συνάρτηση μας επιστρέφει μόνο τα δείγματα και όχι την κλάση τους.

B. Εκπαίδευση Δικτύου

Κατά την διάρκεια της εκπαίδευσης, περνάμε το δείγμα εισόδου στην συνάρτηση που υπολογίζει το σφάλμα. Έτσι "τιμωρούμε" την έξοδο με την είσοδο. Μετά από κάθε εποχή, αποθηκεύουμε το εκάστοτε train_loss στην μεταβλητή train_loss. Έπειτα καλούμε την συνάρτηση test_network για τα δείγματα ελέγχου.

C. Έλεγχος Δικτύου

Στην συνάρτηση test_network() πραγματοποιούμε τον έλεγχο του δικτύου. Για κάθε δείγμα ελέγχου, καλούμε τις

συναρτήσεις feed_forward() και compute_error_o(), προκειμένου να υπολογίσουμε το loss του κάθε δείγματος. Μία παράμετρος της συνάρτησης αυτής είναι το string keyword με βάση το οποίο ξεχωρίζουμε αν πρόκειται για επαλήθευση ή έλεγχο. Όταν πρόκειται για έλεγχο αποθηκεύουμε το μοντέλο στην μεταβλητή trained_model ενώ αν βρισκόμαστε σε κατάσταση επαλήθευσης, υπολογίζουμε και αποθηκεύουμε και το validation loss.

D. Διαγράμματα

Για το reconstruction έχουμε μόνο ένα διάγραμμα που απεικονίζει τα training και validation loss κατά το πέρασμα των εποχών. Αυτό γίνεται από την συνάρτηση plot_loss() η οποία παίρνει απλά ως όρισμα τον αριθμό των εποχών.

E. Δεδομένα με Θόρυβο

- Προκειμένου να παρατηρήσουμε την λειτουργία του Autoencoder με δεδομένα με θόρυβο, δημιουργήσαμε την συνάρτηση add_gaussian_noise(). Η συνάρτηση αυτή παίρνει ως όρισμα μία εικόνα (δηλαδή έναν numpy πίνακα), την μέση τιμή καθώς και την τυπική απόκλιση της κατανομής.

- Στην συνέχεια χρησιμοποιώντας την συνάρτηση np.random.normal() δημιουργούμε τον θόρυβο ο οποίος είναι numpy πίνακας με μέγεθος ίδιο με της εικόνας.

- Στην συνέχεια υπολογίζουμε την τελική εικόνα προσθέτοντας τον θόρυβο στην εικόνα. Επίσης καθώς θέλουμε τα δεδομένα μας να είναι από το 0 έως το 1 χρησιμοποιούμε την συνάρτηση np.clip() προκειμένου η ελάχιστη τιμή θορύβου να είναι το 0 και η μέγιστη το 1.

- Την συνάρτηση αυτή την εφαρμόσαμε στα δείγματα ελέγχου προτού τα περάσουμε από τον Autoencoder.

F. Ανακατασκευή Δεδομένων με *PCA*

Όπως αναφέρεται στην εκφώνηση, στην εργασία αυτή πρέπει να συγκρίνουμε και το reconstrunction του Autoencoder μας με αυτό μέσω PCA. Έτσι χρησιμοποιήσαμε έτοιμες συναρτήσεις PCA προκειμένου να κρατήσουμε ένα συγκεκριμένο ποσοστό των δεδομένων μας. Δημιουργήσαμε μάλιστα την συνάρτηση pca_mnist() η οποία παίρνει ως όρισμα τα δεδομένα.

- Αρχικά καλούμε την συνάρτηση sklearn.decomposition.PCA() προκειμένου να δημιουργήσουμε το αντικείμενο pca.

- Καλούμε έπειτα την συνάρτηση fit() με όρισμα τα δεδομένα μας προκειμένου να υπολογιστούν τα principal components.

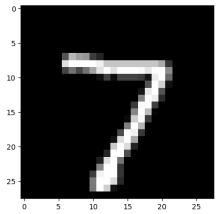
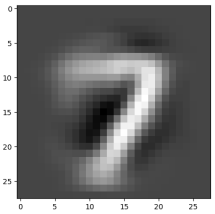
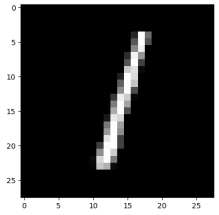
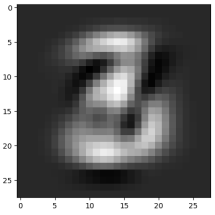
- Η συνάρτηση αυτή πραγματοποιεί center στα δεδομένα οπότε κρατάμε την μέση τιμή των δεδομένων και στην συνέχεια την προσθέτουμε στα δεδομένα μας.

- Προκειμένου να καταλάβουμε πόσα principal components πρέπει να κρατήσουμε, επιλέγουμε αρχικά ποιο ποσοστό της πληροφορίας θέλουμε να κρατήσουμε.

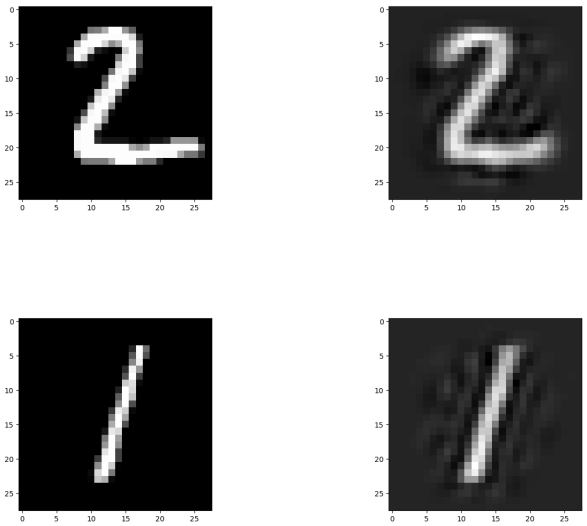
- Το ποσοστό αυτό χρησιμοποιείται καθώς στην μεταβλητή explained_variance_ratio_ είναι αποθηκευμένες οι τιμές που δείχνουν την ποσοστιαία συμβολή κάθε principal component στην διακύμανση των δεδομένων. Έτσι καλώντας και την συνάρτηση cumsum() μπορούμε να βρούμε πόσα principal components χρειάζονται για να πετύχουμε το ζητούμενο ποσοστό.
- Καθώς μειώνονται οι διαστάσεις των δεδομένων και εμείς θέλουμε να κρατήσουμε συγκεκριμένα principal components αλλά να ανακατασκευάσουμε την εικόνα, χρειάζεται να κρατήσουμε τα components που θέλουμε και να εφαρμόσουμε εσωτερικό γινόμενο.
- Το δεύτερο μέλος του εσωτερικού γινομένου είναι η μεταβλητή pca.components[:nComp, :] από την οποία κρατάμε μόνο τα components που μας ενδιαφέρουν. Ουσιαστικά στην μεταβλητή αυτή, η οποία είναι ένας πίνακας nComp * num_of_features, βρίσκεται η πληροφορία της σχέση κάθε feature με κάθε principal component.

Παρακάτω παρατίθενται παραδείγματα εικόνων πριν και μετά το PCA:

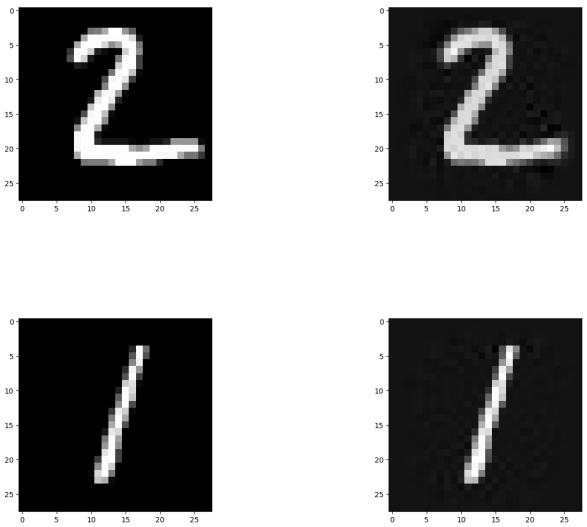
- Κρατώντας το 50 % της πληροφορίας δηλαδή 11 components:

- Κρατώντας το 90 % της πληροφορίας δηλαδή τα 84 principal components :



- Κρατώντας το 99 % της πληροφορίας δηλαδή τα 323 principal components:



1) Παρατηρήσεις

Από τις παραπάνω εικόνες, γίνεται εμφανές πώς με την αύξηση των χαρακτηριστικών που κρατάμε από κάθε εικόνα, αυτές φαίνονται πιο καθαρές. Αυτό ήταν προφανώς αναμενόμενο. Καθώς εδώ η μάθηση δεν είναι επιβλεπόμενη, το αποτέλεσμα είναι απλά πιο θολό, και δεν θυμίζει κάποιον άλλον αριθμό λόγω λάθους, όπως θα παρατηρήσουμε αργότερα στον αυτοκωδικοποιητή.

IV. Πειράματα Ανακατασκευής

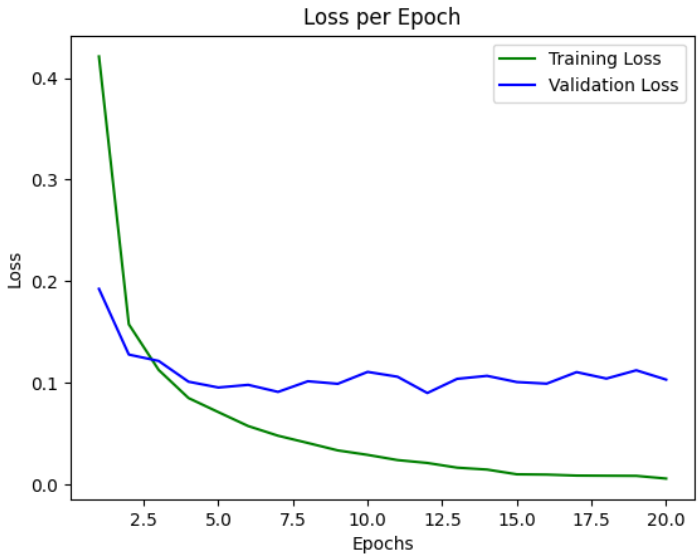
A. Κατηγοριοποιητής

Προκειμένου να ελέγξουμε τις εκόνες που ανακατασκευάζει το δίκτυο μας, χρησιμοποιήσαμε τον classifier της πρώτης εργασίας. Συγκεκριμένα εκπαιδεύσαμε το δίκτυο του classifier με τα εξής χαρακτηριστικά:

Layers	Learning Rate	Epochs
[128, 64, 36, 18]	0.001	20

Όσον αφορά τα training και validation loss καθώς και τα αντίστοιχα accuracies, απεικονίζονται στα παρακάτω διαγράμματα:

- Loss



- Accuracy

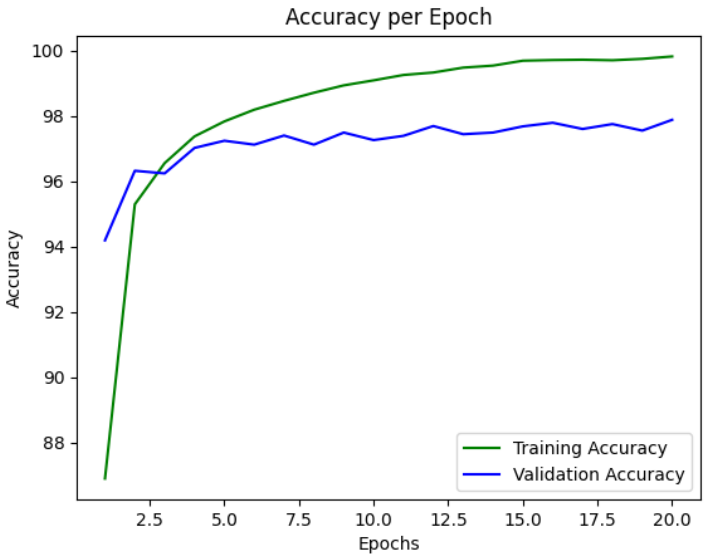


Figure 1: Testing Accuracy: 97.89 %

Παρατηρούμε ότι το δίκτυο έχει εκπαιδευτεί ικανοποιητικά καλά, γεγονός που φαίνεται από τα validation loss και accuracy τα οποία ήδη έχουν αρχισει να αυξομειώνονται. Εξάλλου πρόκειται για εύκολα δεδομένα συγκριτικά με την

CIFAR-10 των προηγούμενων εργασιών.

Χρησιμοποιήσαμε λοιπόν το δίκτυο αυτό προκειμένου να ελέγξουμε τον Autoencoder, και εκπαιδεύσαμε 2 δίκτυα με διαφορετικά layers.

B. Πρώτος *Autoencoder*

Τα χαρακτηριστικά του φαίνονται παρακάτω:

Layers	Learning Rate	Epochs
[128, 64, 36, 18, 9, 18, 36, 64, 128]	0.001	20

Παρακάτω φαίνεται το loss του συγκεκριμένου Autoencoder:

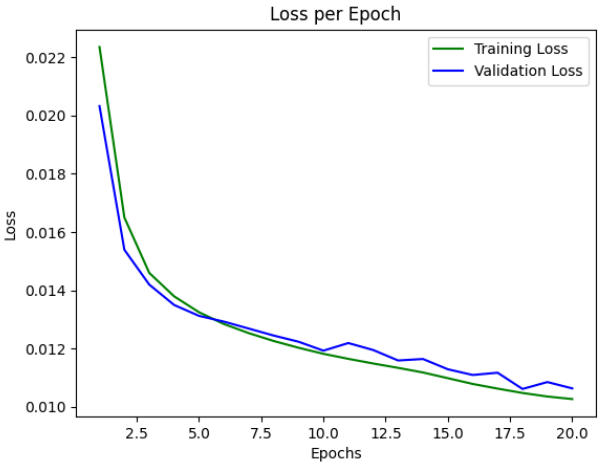
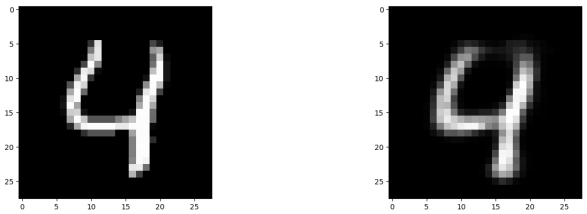


Figure 2: Testing Loss: 0.0112

Παρατηρούμε ότι και τα 2 losses συνεχίζουν να μειώνονται και δεν συγκλίνουν, όμως αρχικά σταματήσαμε εκεί την εκπαίδευση γιατί προσπαθήσαμε να βρούμε ένα καλύτερο δίκτυο. Το δίκτυο αυτό πέτυχε μόλις 91.95 % test accuracy στον classifier που προαναφέραμε.

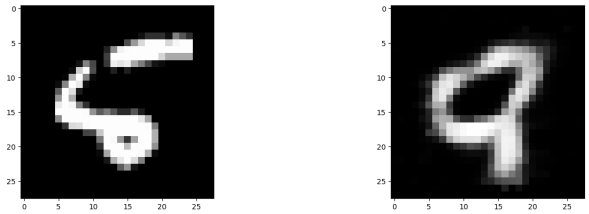
Θα παραθέσουμε κάποιες φωτογραφίες, original και reconstructed προκειμένου να επισημάνουμε τις αδυναμίες του.

- Πρώτη Φωτογραφία:



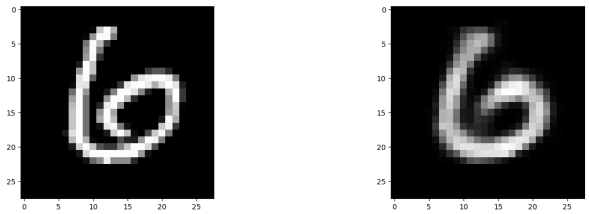
Παρατηρούμε ότι το δίκτυο κατάφερε να ανακατασκευάσει το 4 σε ικανοποιητικό βαθμό.

- Δεύτερη Φωτογραφία:



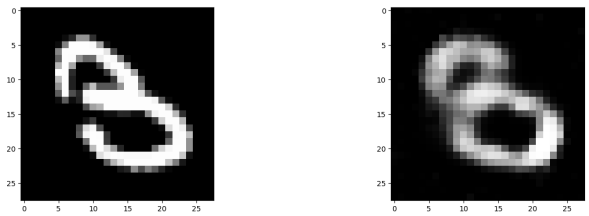
Είναι φανερό πως το δίκτυο απέτυχε να αναγνωρίσει τα μοτίβα στο συγκεκριμένο 6 το οποίο είναι και λογικό καθώς είναι αρκετά περίεργο 6.

- Τρίτη Φωτογραφία:



Το δίκτυο τα κατάφερε στο συγκεκριμένο 6 καθώς ήταν πιο εύκολο και ίσως πιο ξεκάθαρο.

- Τέταρτη Φωτογραφία:



Το 3 της παραπάνω εικόνας, ανακατασκευάστηκε ως 8, γεγονός που φαντάζει λογικό καθώς το συγκεκριμένο 3 είναι αρκετά κοντά στο 8. Είναι εμφανές ότι ένας σωστός κατηγοριοποιητής θα έβαζε το reconstructed 3 σε μία λάθος κατηγορία.

C. Δεύτερος *Autoencoder*

Δοκιμάσαμε να αυξήσουμε τις διαστάσεις στο bottle neck layer καθώς και στα άλλα layers προκειμένου να παρατηρήσουμε την συμπεριφορά του δικτύου.

Τα χαρακτηριστικά του νέου δικτύου αυτοκωδικοποίησης είναι:

Layers	Learning Rate	Epochs
[256, 128, 64, 128, 256]	0.001	50

Τα training και validation loss στην διάρκεια των 50 εποχών φαίνονται στο παρακάτω διάγραμμα:

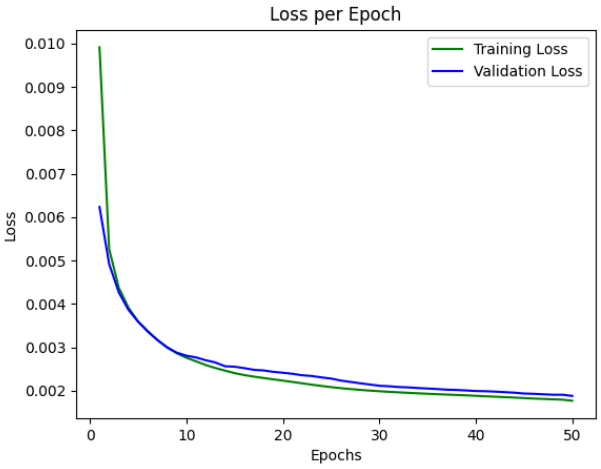
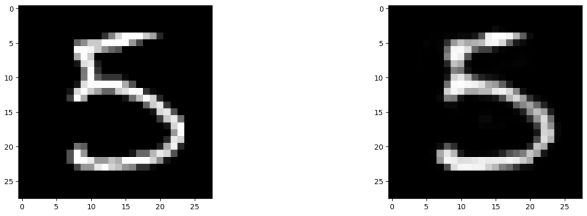


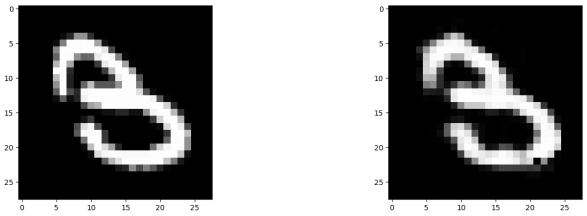
Figure 3: Testing Loss: 0.00188

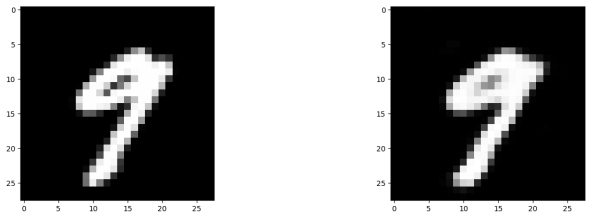
- Παρατηρούμε ότι τα losses έχουν μειωθεί κατά πολύ και έτσι αναμένουμε οι reconstructed φωτογραφίες του δικτύου να αποφέρουν καλύτερα αποτελέσματα στον classifier καθώς και αισθητή διαφορά στο μάτι.

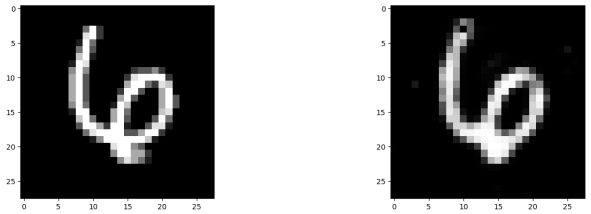
- Περνώντας λοιπόν τις reconstructed φωτογραφίες στον classifier πετύχαμε testing accuracy 97.58 %. Αυτό είναι πολύ καλύτερο από του πρώτου Autoencoder και πολύ κοντά σε αυτό των original φωτογραφιών, το οποίο είναι 97.89 %.

Παραθέτουμε έτσι κάποιες φωτογραφίες του νέου δικτύου:









- Παρατηρούμε ότι προφανώς το δίκτυο αυτό ανακατασκευάζει καλύτερα τις εικόνες. Συγκεκριμένα το 3 ανακατασκευάζεται ως 3 και όχι ως 8 και ίσως είναι και πιο ξεκάθαρο 3 από ότι η αρχική εικόνα. Επίσης το 6 ανακατασκευάζεται πολύ σωστά παρότι είναι δύσκολο.

D. Φωτογραφίες *PCA*

Από τον κατηγοριοποιητή περάσαμε και εικόνες που αναπαρήγαγε το PCA. Παρακάτω φαίνονται τα αντίστοιχα ποσοστά επιτυχίας:

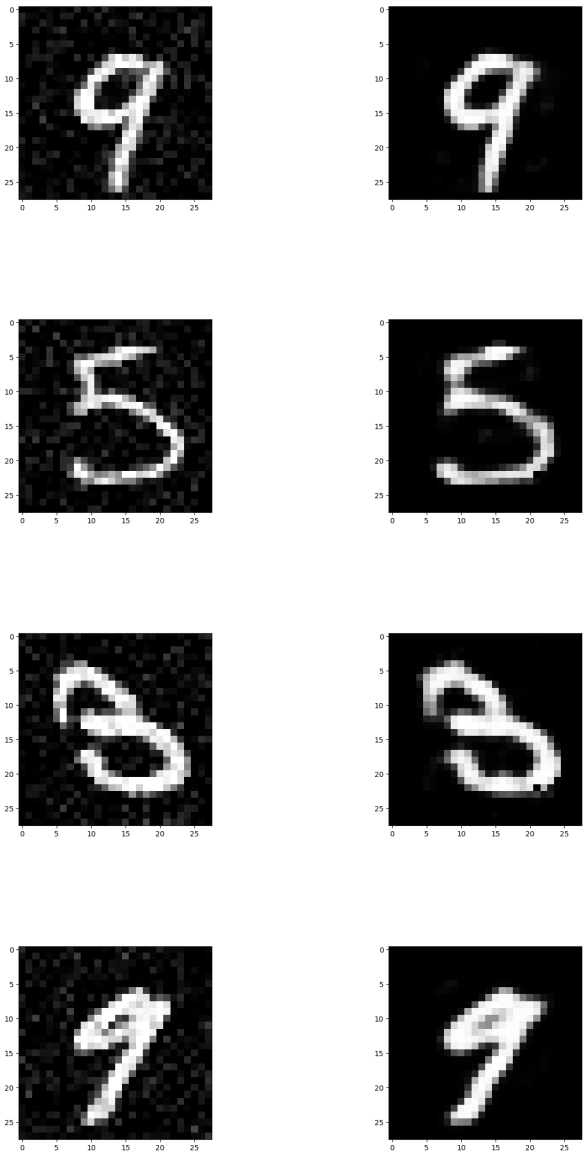
PCA	Ακρίβεια Ελέγχου
50 %	86.04 %
90 %	97.75 %
99 %	97.9 %

Παρατηρούμε ότι οι εικόνες που ανακατασκευάζει το PCA, είναι εύκολα αναγνωρίσιμες από τον κατηγοριοποιητή. Ακόμα και όταν κρατάμε 50 % της πληροφορίας καταλαβαίνει να το αναγνωρίζει ενώ το 90 % με το 99% έχουν πολύ μικρή διαφορά ακρίβειας. Γενικά το PCA καθώς δεν εκπαιδεύεται με επιβλεπόμενη μάθηση, απλά εμφανίζει πιο θολό το αποτέλεσμα όταν επιλεχθεί χαμηλή τιμή ακρίβειας. Ωστόσο ο autoencoder μπορεί πολλές φορές να μπερδέψει ακόμα και το νούμερο, αν αυτό δεν είναι ξεκάθαρο.

E. Φωτογραφίες με Θόρυβο

Καθώς είναι γνωστό ότι η χρήση Autoencoder σε δεδομένα με θόρυβο, τον απομακρύνει, δοκιμάσαμε στο δεύτερο δίκτυο Autoencoder να ελέγξουμε το εκπαιδευμένο μοντέλο με noisy δεδομένα.

Παρακάτω παραθέτουμε μερικές από τις φωτογραφίες με θόρυβο και τις αντίστοιχες ανακατασκευασμένες:



Παρατηρούμε ότι το Δίκτυο Αυτοκωδικοποίησης, όχι μόνο απομακρύνει τον θόρυβο αλλά ανακατασκευάζει με επιτυχία τις φωτογραφίες.

V. Επόμενο Ψηφίο

Στο επόμενο μέρος της εργασίας, τροποποιήσαμε τον κώδικα προκειμένου να κατασκευάσουμε το επόμενο ψηφίο.

- Για να το πετύχουμε αυτό, δίνουμε ως είσοδο στον αυτοκωδικοποιητή αριθμούς από το 0 έως το 9 και υπολογίζουμε το σφάλμα της εξόδου με βάση το επόμενο ψηφίο.

- Για την επιλογή του επόμενου ψηφίου, καταλήξαμε να έχουμε μια προκαθορισμένη λίστα με μία φωτογραφία από κάθε αριθμό. Αυτό συνέβη καθώς όταν το δίκτυο προσπαθούσε να βγάλει διαφορετικούς επόμενους αριθμούς (π.χ. διαφορετικά 4 να βγάλουν διαφορετικές φωτογραφίες του 5), το αποτέλεσμα ήταν θολό, καθώς ήταν μία προσπάθεια του δικτύου να αποτυπώσει τον μέσο όρο των εικόνων.

- Μία άλλη προσέγγιση που ακολουθήθηκε είναι να μάθουμε στο δίκτυο να αναπαράγει τον εαυτό του, αλλά χρησιμοποιώντας ενδιάμεσα μία softmax προκειμένου να αναγνωρίζεται το ψηφίο. Γενικά η χρήση softmax αποκλίνει από την λογική του αυτοκωδικοποιητή όμως καθώς δοκιμάστηκε, το συμπεριλάβαμε και αυτό. Παρακάτω θα παρουσιαστεί αναλυτικότερα ο κώδικας.

A. Πρώτη Προσέγγιση

Στην πρώτη προσέγγιση που αναφέρθηκε, δεν χρησιμοποιούμε softmax και η δομή του κώδικα είναι ίδια με την ανακατασκευή ψηφίου. Ωστόσο υπάρχουν διαφοροποιήσεις στις συναρτήσεις train και test καθώς και στην επεξεργασία των αρχικών δεδομένων. Ο τροποποιημένος κώδικας βρίσκεται στο αρχείο auteoncoder_next_digit.py

Τα χαρακτηριστικά του δικτύου είναι:

Layers	Learning Rate	Epochs
[256, 128, 64, 128, 256]	0.001	30

1) Επεξεργασία Δεδομένων

- Προκειμένου να επεξεργαστούμε τα δεδομένα που έχουμε φορτώσει με την load_mnist(), δημιουργούμε την συνάρτηση seperate_digits(). Αυτή παίρνει ως παραμέτρους δείγματα και τις αντίστοιχες κλάσεις τους, και επιστρέφει μία λίστα μεγέθους 10, το κάθε στοιχείο της οποίας είναι λίστα δειγμάτων που ανήκουν στην ίδια κλάση. Έτσι χωρίζουμε τα δείγματα εκπαίδευσης και ελέγχου ανάλογα με τις κλάσεις τους καλώντας 2 φορές αυτή την συνάρτηση.

- Ακόμη δημιουργούμε την συνάρτηση expected_output(), η οποία παίρνει ως όρισμα την λίστα από λίστες που δημιουργήσαμε, και επιστρέφει μία λίστα με 10 στοιχεία. Το κάθε στοιχείο είναι ενας numpy πίνακας μεγέθους 784. Συγκεκριμένα η εικόνα στην θέση 0 είναι μία εικόνα του αιρθμού 1 που είναι ο επόμενος αριθμός του μηδενός. Αντίστοιχα το στοιχείο στη θέση 9 είναι μία εικόνα του αριθμού 0.

- Έτσι στις συναρτήσεις train_network() και test_network() περνάμε τα χωρισμένα πλέον δεδομένα, χωρίς τις κλάσεις τους, μαζί με τα αναμενόμενα ψηφία που θέλουμε να παράγει το δίκτυο.

2) Δεδομένα Εκπαίδευσης

- Προκειμένου το δίκτυο να εκπαιδευτεί σωστά, δεν πρέπει να εκπαιδευτεί βλέποντας τους αριθμούς με την σειρά, δηλαδή πρώτα όλα τα μηδενικά και μετά όλους τους άσους κοκ. Έτσι για κάθε αριθμό δημιουργούμε έναν iterator χρησιμοποιώντας την συνάρτηση cycle().

- Έπειτα χρησιμοποιούμε μία for στην οποία εξετάζονται κυκλικά όλοι οι αριθμοί και όλα τα δείγματα από κάθε αριθμό. Αυτό γίνεται για 10 φορές επί τον μέγιστο αριθμό εμφάνισης των αριθμών. Έτσι κάποιοι αριθμοί εμφανίζονται παραπάνω φορές ενώ εξετάζονται όλα τα δείγματα του αριθμού με τις πιο πολλές εμφανίσεις.

- Τα αποτελέσματα από αυτή την προσέγγιση ήταν ιδιαίτερα ικανοποιητικά, με το loss να είναι αρκετά μικρό τόσο στην εκπαίδευση όσο και στην επαλήθευση.

3) Αποτελέσματα

Τα training και validation loss με το πέρασμα των εποχών φαίνονται παρακάτω:

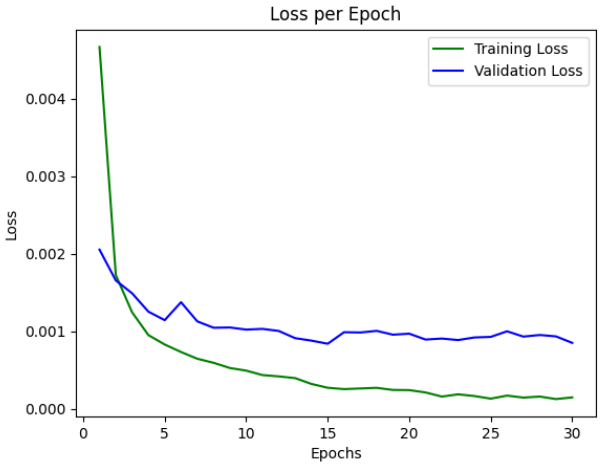
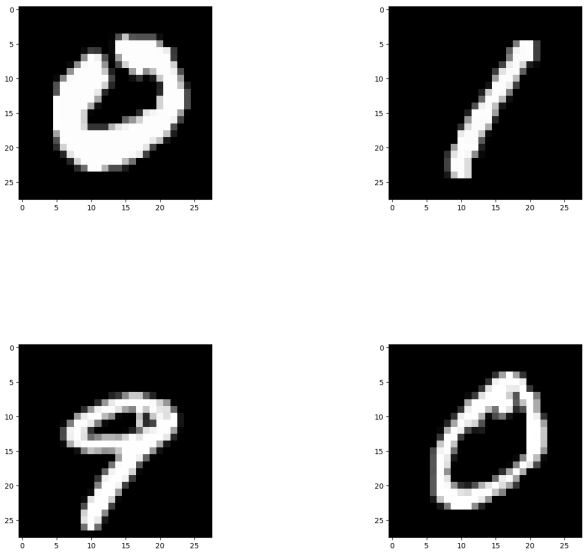


Figure 4: Testing Loss: 0.0008

- Μερικά παραδείγματα κατασκευής επόμενο ψηφίου παρατίθενται παρακάτω:



- Τα αποτελέσματα είναι πολύ καθαρά, γεγονός που αναμέναμε, καθώς μάθαμε στο δίκτυο να βγάζει μία συγκεκριμένη εικόνα για κάθε αριθμό.

B. Δεύτερη Προσέγγιση

Ο κώδικας της προσέγγισης αυτής βρίσκεται στο αρχείο autoencoder_next_digit_softmax.py

- Στην προσέγγιση αυτή, χρησιμοποιήσαμε softmax στο bottle-neck layer και τροποποιήσαμε τον κώδικα προκειμένου να εφαρμόσουμε σωστά τις αλλαγές. Ουσιαστικά ο encoder λειτουργεί ως classifier, και εκπαιδεύεται με cross entropy loss και softmax.

- Στην διάρκεια της εκπαίδευσης η είσοδος του decoder είναι το ground truth one hot encoded vector. Ωστόσο στην διάρκεια του ελέγχου, ζητάμε από τον decoder να εμφανίσει το επόμενο ψηφίο από αυτό που προέβλεψε ο decoder.

- Για παράδειγμα στην διάρκεια της εκπαίδευσης αν η είσοδος δείχνει το 3, θα περάσουμε την τιμή 3 ως one hot encoded στον decoder, ασχέτως της πρόβλεψης του encoder. Στην διάρκεια του ελέγχου, αν η έξοδος του encoder δίνει 3, περνάμε την τιμή 4 ως one hot encoded vector στον decoder.

- Για τον σκοπό αυτό χρησιμοποιήσαμε την συνάρτηση one_hot_encoded_num(), η οποία παίρνει ως όρισμα έναν αριθμό και επιστρέφει το αντίστοιχο one hot encoded numpy πίνακα.

Τα χαρακτηριστικά του δικτύου είναι:

Layers	Learning Rate	Epochs
[128, 64, 36, 10, 36, 64, 128]	0.001	30

- Χωρίσαμε την συνάρτηση feed_forward() σε feed_forward_encoder() και feed_forward_decoder(). Η πρώτη παίρνει τώρα ως όρισμα το δείγμα το οποίο περνάμε από το δίκτυο, ενώ η δεύτερη την είσοδο που ορίζουμε ως decoder_input.

- Καθώς υπολογίζουμε 2 διαφορετικά errors με τα οποία εκπαιδεύουμε το δίκτυο, χωρίζουμε την συνάρτηση computer_error_o() σε compute_error_encoder() και compute_error_decoder(). Η πρώτη παίρνει ως όρισμα το target output που είναι η πραγματική κλάση του δείγματος, one hot encoded, ενώ η δεύτερη το target output του αυτοκωδικοποιητή που είναι ενας numpy πίνακας μεγέθους 784.

- Αντίστοιχα χωρίζουμε και το back_propagation(), και μάλιστα πλέον ο encoder και decoder έχουν διαφορετική λίστα με delta. Το delta output του decoder είναι:

$$\delta_j = e(j) * \varphi'_j(u_j)$$

Το delta output του encoder είναι:

$$\delta_j = e(j)$$

1) Διαγράμματα

Κατά τη διάρκεια του training και του validation αποθηκεύσαμε τα accuracies του encoder καθώς και τα encoder και decoder loss. Τα αντίστοιχα διαγράμματα φαίνονται παρακάτω:

- Encoder Accuracy:

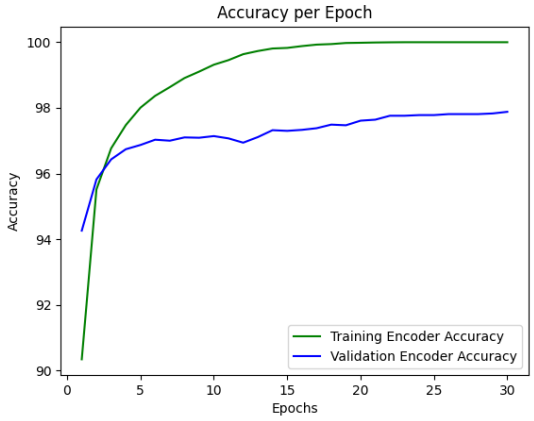


Figure 5: Testing Accuracy: 97.88 %

- Encoder Loss:

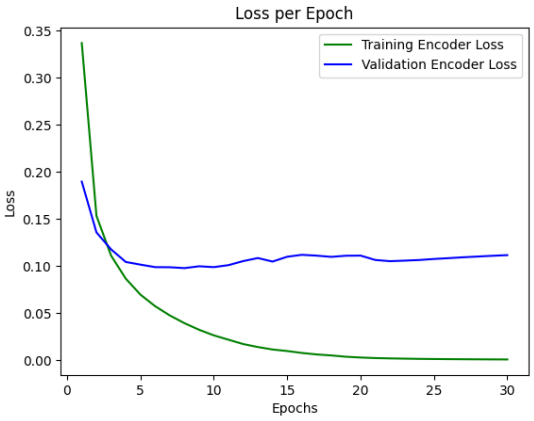
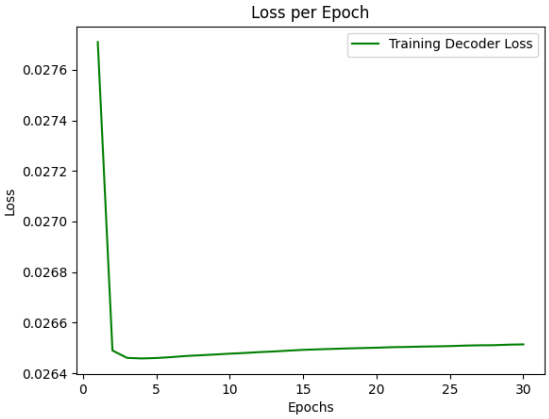


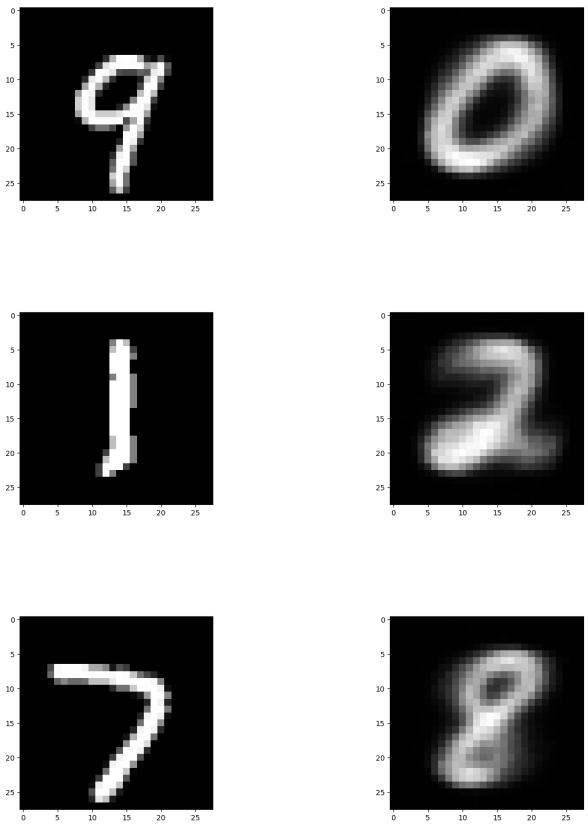
Figure 6: Testing Loss: 0.11

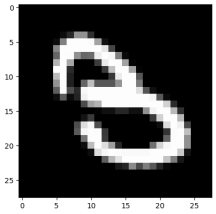
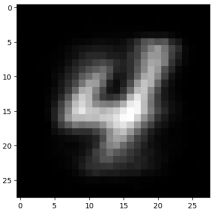
- Decoder Loss:



Παρατηρούμε ότι ο classifier πετυχαίνει πολύ καλά ποσοστά επιτυχίας. Αυτό ήταν αναμενόμενο καθώς χρησιμοποιήσαμε επαρκή layers και το σύνολο δεδομένων της MNIST είναι αρκετά εύκολο στην κατηγοριοποίηση. Ακόμη ο decoder έχει σχετικά μικρό loss αλλά φαίνεται να αυξάνεται ελάχιστα με τις εποχές. Το γεγονός ότι δίνουμε στον decoder απλά ένα one hot encoded vector και ότι προσπαθούμε κάθε φορά να του μάθουμε να βγάζει διαφορετικές εικόνες από το ίδιο νούμερο, προκαλεί αυτό το loss.

- Παρακάτω φαίνονται μερικές εικόνες πριν και μετά την είσοδο στο δίκτυο:



- Όπως φαίνεται από τις παραπάνω εικόνες, το δίκτυο αναγνωρίζει το νούμερο και εμφανίζει με επιτυχία το επόμενο αλλά αυτό είναι θολό.

- Να σημειωθεί ότι δεν υπολογίσαμε το validation decoder loss εφόσον δεν μπορούσαμε να συγκρίνουμε το output με κάποια συγκεκριμένη εικόνα, ώστε να υπάρχει νόημα για το loss.

VI. Αθροιστής

Σε αυτό το κομμάτι της εργασίας ασχοληθήκαμε με την δημιουργία ενός αθροιστή ψηφίων της MNIST, χρησιμοποιώντας το δίκτυο αυτοκωδικοποίησης. Προκειμένου να πετύχουμε το ζητούμενο, συνδυάσαμε την λογική της κατασκευής του επόμενου ψηφίου με αυτήν της ανακατασκευής.

- Προκειμένου το δίκτυο να παράγει 2 εικόνες στην έξοδο οι οποίες είναι το άθροισμα των εικόνων εισόδου, μία λύση είναι να εκπαιδευθεί να αναγνωρίζει τις εικόνες και να αθροίζει τις κλάσεις που έχει προβλέψει. Έτσι με παρόμοια λογική με αυτή του επόμενου ψηφίου με χρήση softmax, μπορεί να παραχθεί το ζητούμενο.

- Μία άλλη λογική, είναι να εκπαιδεύσουμε το δίκτυο, έτσι ώστε όταν βλέπει στην είσοδο έναν συνδυασμό από νούμερα, να βγάζει κάποιες αναμενόμενες φωτογραφίες. Δηλαδή να υπάρχει μία λίστα από 10 αναμενόμενα ψηφία, και ο αυτοκωδικοποιητής απλά να μάθει ανάλογα με τα χαρακτηριστικά των εικόνων εισόδου, να βγάζει έναν συνδυασμό 2 εικόνων από την λίστα με τα αναμενόμενα ψηφία.

A. Επεξεργασία Δεδομένων

Προκειμένου να περάσουμε τα δεδομένα μας από τον αυτοκωδικοποιητή, δημιουργούμε την συνάρτηση create_input_2().

- Η συνάρτηση αυτή έχει ως παραμέτρους τα δεδομένα και τις αντίστοιχες κλάσεις τους.

- Δημιουργούμε 1 λίστα οι οποία περιέχοει ζευγάρια εικόνων. Αυτά δημιουργούνται έτσι ώστε η κάθε εικόνα να συμμετέχει σε δύο ζευγάρια. Ουσιαστικά το κάθε ζευγάρι είναι ένα numpy πίνακας μεγέθους 1568 x 1 και είναι το αποτέλεσμα του np.concatenate() των 2 εικόνων. Ταυτόχρονα, αποθηκεύουμε τις κλάσεις και των δύο εικόνων σε μία λίστα από tuples.

- Η συνάρτηση επιστρέφει τα δεδομένα και τις κλάσεις και των 2 εικόνων του κάθε ζευγαριού.

- Ακόμη δημιουργούμε και την συνάρτηση create_specific_digits() η οποία επιστρέφει μία λίστα με μία εικόνα για κάθε αριθμό απο το 0 έως το 9. Αυτή περνάει ως όρισμα στην συνάρτηση train_network().

B. Πρώτη Προσέγγιση

Στην υλοποίηση αυτή, δεν χρησιμοποιούμε softmax και έτσι χρησιμοποιούμε τον κώδικα που δεν έχει χωρισμένες τις συναρτήσεις σε encoder και decoder. Ο κώδικας μπορεί να βρεθεί στο αρχείο autoencoder_adder.py

Τα χαρακτηριστικά του δικτύου είναι:

Layers	Learning Rate	Epochs
[256, 128, 64, 128, 256]	0.001	30

1) Διαδικασία Εκπαίδευσης

- Στην συνάρτηση train_network() πραγματοποιούμε την εκπαίδευση του δικτύου. Αρχικά καλούμε την συνάρτηση feed_forward() στην οποία περνάμε το δείγμα μας με τις 2 εικόνες.

- Στην συνέχεια καλούμε την συνάρτηση seperate_digits() για το άθροισμα των κλάσεων των 2 εικόνων. Η συνάρτηση αυτή μετατρέπει τον ακέραιο αριθμό που της δίνεται, σε string. Έπειτα παίρνει το πρώτο και το δεύτερο στοιχείο του string και τα επιστρέφει ως ακεραίους. Έτσι ξέρουμε ποιες εικόνες πρέπει να εμφανιστούν στην έξοδο σύμφωνα με την αντίστοιχη είσοδο. Σε περίπτωση που το άθροισμα είναι μονοψήφιος αριθμός, προστίθεται στην αρχή του string το 0, προκειμένου να λειτουργήσει σωστά ο κώδικας.

- Έπειτα κατασκευάζουμε την επιθυμητή έξοδο, πραγματοποιώντας concatenate μεταξύ των αντίστοιχων φωτογραφιών της λίστας expected_nums(), τις οποίες θέλουμε να εμφανίζει το δίκτυο.

- Προφανώς χρησιμοποιούμε Mean Squared Error ως συνάρτηση loss και έτσι προσαρμόζουμε τις εξισώσεις. Παρόμοια διαδικασία εφαρμόζουμε και κατά την διάρκεια του ελέγχου.

- Κατά την διάρκεια του ελέγχου αποθηκεύουμε την έξοδο του δικτύου καθώς και τις αντίστοιχες εισόδους.

- Στην συνάρτηση plot_loss() χρησιμοποιώντας συναρτήσεις της matplotlib.pyplot, τυπώνουμε διαγράμματα που δείχνουν το loss στην διάρκεια της εκπαίδευσης και της επαλήθευσης.

Layers	Learning Rate	Epochs
[256, 128, 64, 128, 256]	0.001	30

Τα χαρακτηριστικά του δικτύου είναι τα εξής:
Παρακάτω παρατίθεται το διάγραμμα του loss:

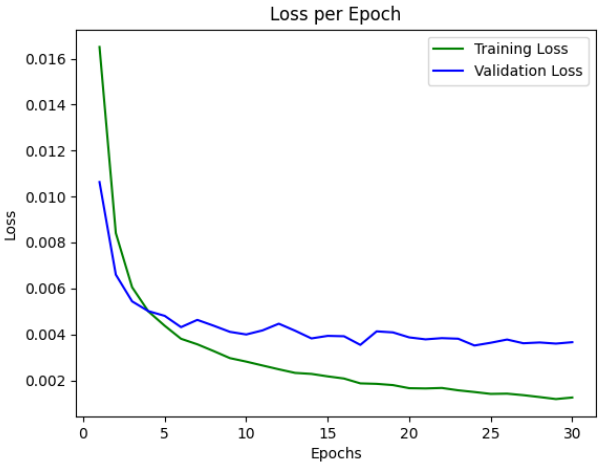
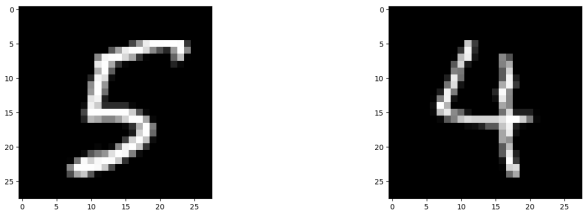
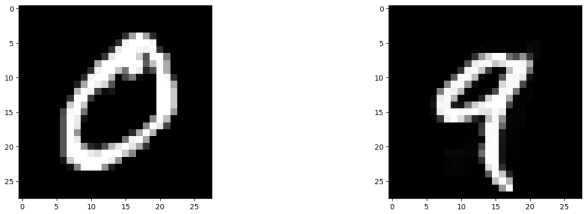


Figure 7: Testing Loss: 0.0036

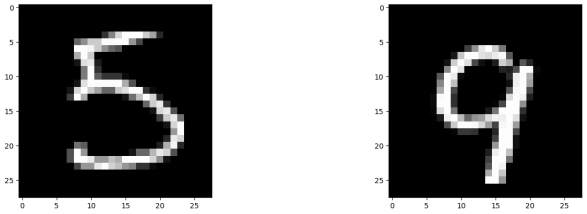
Ακόμη θα παραθέσουμε κάποιες φωτογραφίες που άθροισε το δίκτυο, καθώς και το αποτέλεσμα:

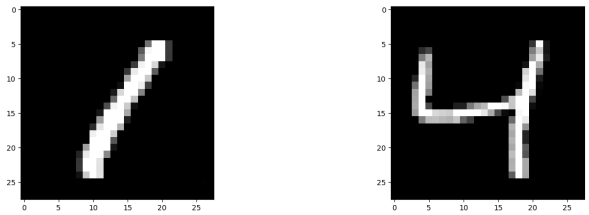
- Πρώτο Άθροισμα:



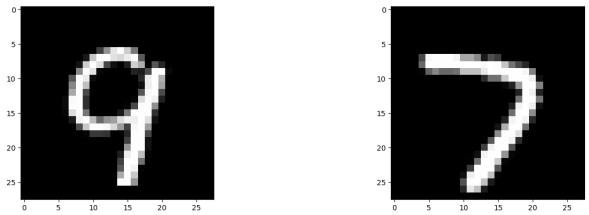


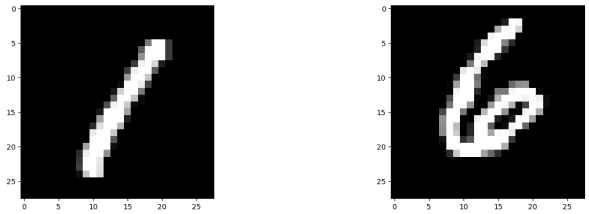
- Δεύτερο Άθροισμα:





- Τρίτο Άθροισμα:





Όπως φαίνεται από το loss το οποίο είναι ιδιαίτερα μικρό, και απο τις φωτογραφίες, το δίκτυο κατάφερε να λειτουργήσει ως αθροιστής και μάλιστα με μεγάλη επιτυχία.

C. Δεύτερη Προσέγγιση

Σε αυτή την υλοποίηση, χρησιμοποιήσαμε τον κώδικα με τις ξεχωριστές συναρτήσεις για encoder και decoder. Ο τροποποιημένος κώδικας βρίσκεται στο αρχείο autoencoder_adder_softmax.py. Παρότι θα μπορούσαμε να χρησιμοποιήσουμε ακριβώς την υλοποίηση για το επόμενο ψηφίο με softmax, και να περνάμε την κάθε εικόνα με την σειρά από τον classifier, ακολουθήσαμε μία άλλη λογική. Περνάμε και τις 2 εικόνες μαζί και τις αναγνωρίζουμε ταυτόχρονα με 2 softmax και 20 νευρώνες στο bottle-neck layer. Καθώς ψάχνουμε το ίδιο χαρακτηριστικό, δηλαδή την κλάση των δύο εικόνων, η διαδικασία ίσως είναι περιττή αλλά καθώς υλοποιήθηκε την παρουσιάζουμε.

Τα χαρακτηριστικά του δικτύου είναι:

Layers	Learning Rate	Epochs
[128, 64, 36, 20, 36, 64, 128]	0.001	30

Παρακάτω θα τονίσουμε τις διαφορές από τις προηγούμενες υλοποιήσεις:

- Σε σχέση με τον προηγούμενο adder, εδώ οι κλάσεις των δεδομένων είναι one hot encoded.

- Τα δεδομένα χωρίζονται πάλι σε ζευγάρια προκειμένου να δοθούν στο δίκτυο αυτοκωδικοποίησης.

- Αναγνωρίζουμε κάθε μία από τις φωτογραφίες ξεχωριστά, και έχουμε 20 νευρώνες στο latent vector για αυτό τον σκοπό. Στους 10 πρώτα εφαρμόζεται softmax και τα βάρη που συνδέονται με αυτούς τους νευρώνες ανανεώνονται με βάση αυτά τα gradients, και το αντίστοιχο πραγματοποιείται για τους επόμενους 10 νευρώνες.

- Σε αυτή την υλοποίηση, χρησιμοποιούμε πάλι αναμενόμενες φωτογραφίες. Δηλαδή αν στην διάρκεια του training λάβουμε φωτογραφίες του 5 και του 6, μαθαίνουμε στο δίκτυο να βγάλει συγκεκριμένες φωτογραφίες του 5 και του 6, concatenated.

- Στην διάρκεια του testing περνάμε από τον decoder τα one hot encoded διανύσματα του αθροίσματος των κλάσεων της εισόδου. Δηλαδή αν μας δοθούν οι φωτογραφίες 5 και 6, θα περάσουμε 2 one hot encoded διανύσματα του 1, καθώς το άθροισμα τους είναι 11, και τελικά θα έχουμε 2 εικόνες του 1 στην έξοδο.

- Τελικά υπολογίζουμε τα accuracies και τα loss τόσο για την εκπαίδευση όσο και για την επαλήθευση καθώς και για κάθε ένα από τα 2 ψηφία.

Τα δεδομένα που προαναφέρθηκαν φαίνονται παρακάτω:

- Decoder Loss

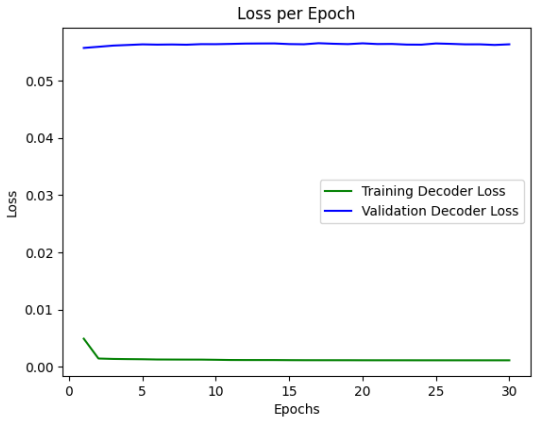


Figure 8: Testing Loss: 0.056

- Encoder First Digit Loss

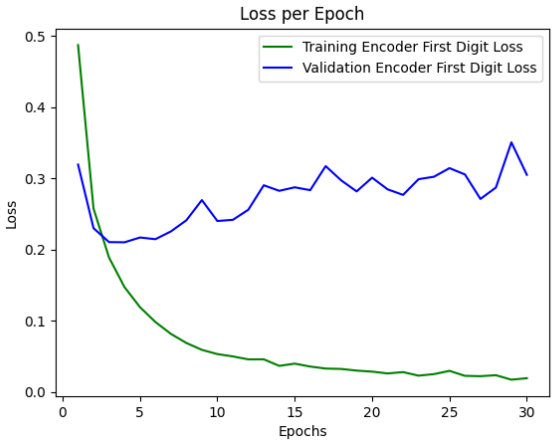


Figure 9: Testing Loss: 0.305

- Encoder Second Digit Loss

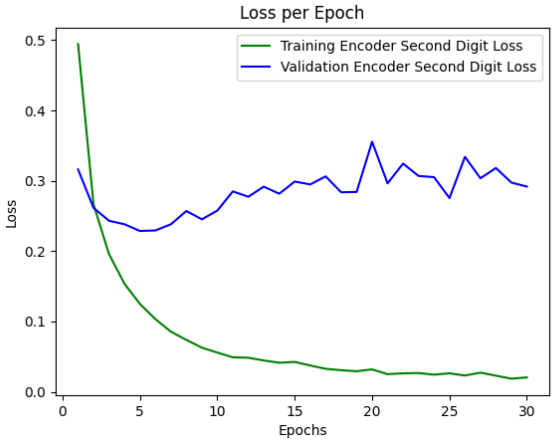


Figure 10: Testing Loss: 0.2917

- Encoder First Digit Accuracy

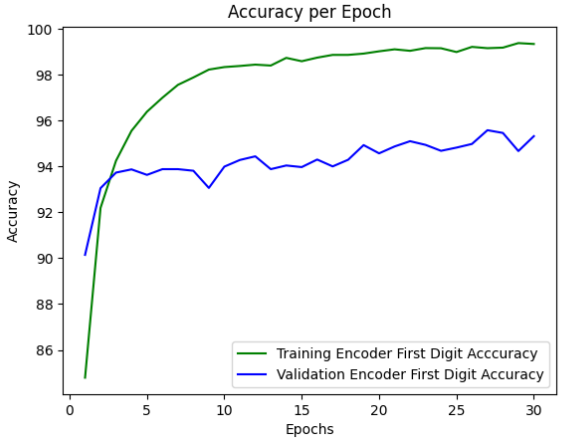


Figure 11: Testing Accuracy: 95.32 %

- Encoder Second Digit Accuracy

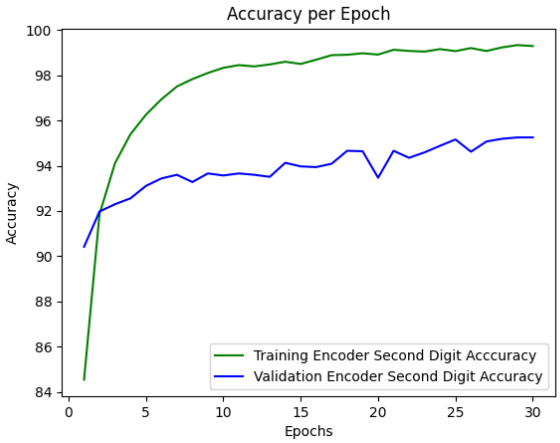
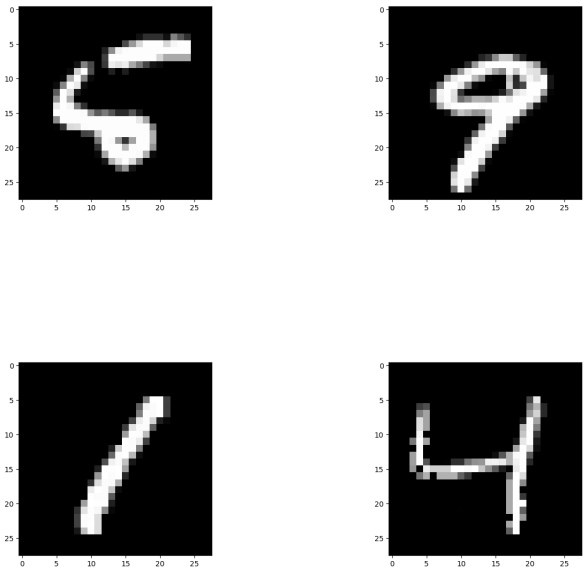


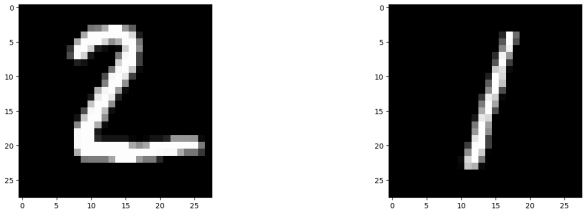
Figure 12: Testing Accuracy: 95.24 %

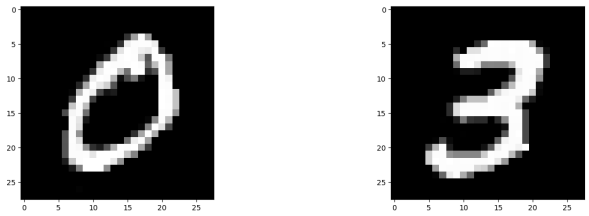
Παρακάτω παρατίθενται εικόνες αθροισμάτων αυτής της υλοποίησης:

- Πρώτο Άθροισμα:

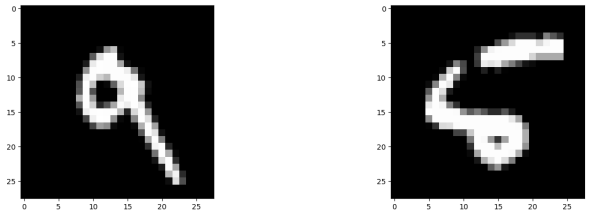


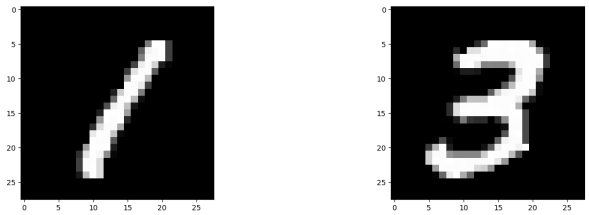
- Δεύτερο Άθροισμα:



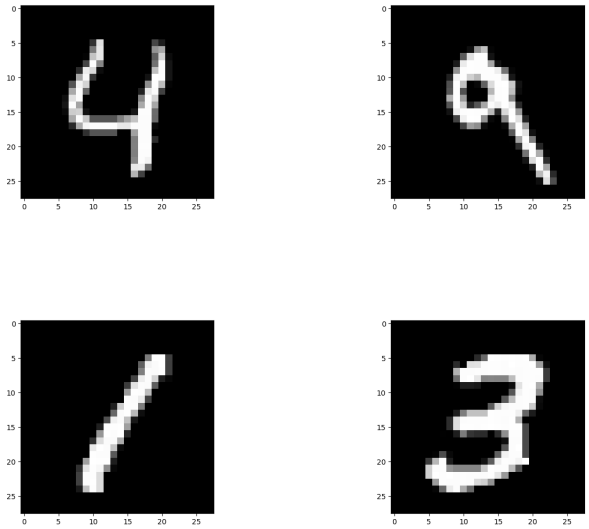


- Τρίτο Άθροισμα:





- Τέταρτο Άθροισμα:



1) Παρατηρήσεις

Όπως φαίνεται από τα παραπάνω διαγράμματα, η ακρίβεια αναγνώρισης ψηφίων είναι μικρότερη σε σχέση με αυτή που επιτυγχάνεται για την αναγνώριση μίας εικόνας. Ακόμη το loss για το κάθε ψηφίο είναι μεγαλύτερο από το αντίστοιχο loss του classifier. Αυτό είναι λογικό και έτσι αναμένουμε

και στην διάρκεια του validation και του testing χαμηλότερες επιδόσεις, αφού πολλές φορές αναγνωρίζονται λανθασμένα τα ψηφία. Ακόμη το test loss του decoder είναι 0.056, ενώ του δικτύου χωρίς softmax είναι 0.0036 δηλαδή πολύ μικρότερο.

VII. Χρόνοι Εκπαίδευσης

Προκειμένου να συγκρίνουμε τους χρόνους εκπαίδευσης των πειραμάτων, παραθέτουμε τον χρόνο εκπαίδευσης για 20 εποχές σε όλα τα δίκτυα:

Ανακατασκευή Ψηφίου	Χρόνος Εκπαίδευσης
Πρώτη Προσέγγιση	17 Λεπτά
Δεύτερη Προσέγγιση	1 ώρα και 20 λεπτά

Ανακατασκευή Επόμενου Ψηφίου	Χρόνος Εκπαίδευσης
Πρώτη Προσέγγιση	1 ώρα και 34 λεπτά
Δεύτερη Προσέγγιση	22 λεπτά

Αθροιστής	Χρόνος Εκπαίδευσης
Πρώτη Προσέγγιση	3 ώρες και 6 λεπτά
Δεύτερη Προσέγγιση	1 ώρα και 14 λεπτά

- Παρατηρούμε ότι αυξάνεται ο χρόνος εκπαίδευσης όσο αυξάνονται οι νευρώνες στα layers. Ειδικά για την πρώτη προσέγγιση του αθροιστή, τόσο η είσοδος όσο και η έξοδος έχουν μέγεθος 1526 και μάλιστα έχει αυξημένο αριθμό νευρώνων και στα κρυφά layers.

- Στην ανακατασκευή ψηφίου και συγκεκριμένα στην πρώτη προσέγγιση, χρησιμοποιούμε λιγότερους νευρώνες και η είσοδος και η έξοδος έχουν μέγεθος 784 γι'αυτό και έχει τον μικρότερο χρόνο.

VIII. Βιβλιογραφία

- https://www.v7labs.com/blog/autoencoders-guide
- https://en.wikipedia.org/wiki/Autoencoder
- https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798
- https://www.analyticsvidhya.com/blog/2021/06/autoencoders-a-gentle-introduction/
- https://www.geeksforgeeks.org/auto-encoders/
- https://shivammehta25.github.io/posts/deriving-categorical-cross-entropy-and-softmax/