Μηχανική Μάθηση

2ο Σετ Ασκήσεων

ΟΝΟΜΑ: Βασίλειος
ΕΠΩΝΥΜΟ: Καλαϊτζόπουλος
ΑΡΙΘΜΟΣ ΜΗΤΡΩΟΥ: 1066670
ΕΤΟΣ ΦΟΙΤΗΣΗΣ: 4ο
ΤΜΗΜΑ: Ηλεκτρολόγων Μηχανικών & Τεχνολογίας Υπολογιστών

Πρόβλημα 2.1

Περιγραφή Προβλήματος

Στο συγκεκριμένο πρόβλημα έχουμε ένα εκπαιδευμένο generative μοντέλο το οποίο δημιουργεί ''χειρόγραφα'' οκτώ όταν στην είσοδο του δέχεται ένα διάνυσμα Z διάστασης 10 x 1 του οποίου τα στοιχεία προέρχονται τυχαία από την gaussian κατανομή με κέντρο 0 και διασπορά 1. Το ''χειρόγραφο'' οκτώ στην έξοδο του είναι ένα διάνυσμα 784 x 1 το οποίο προκύπτει από τις παρακάτω εξισώσεις:

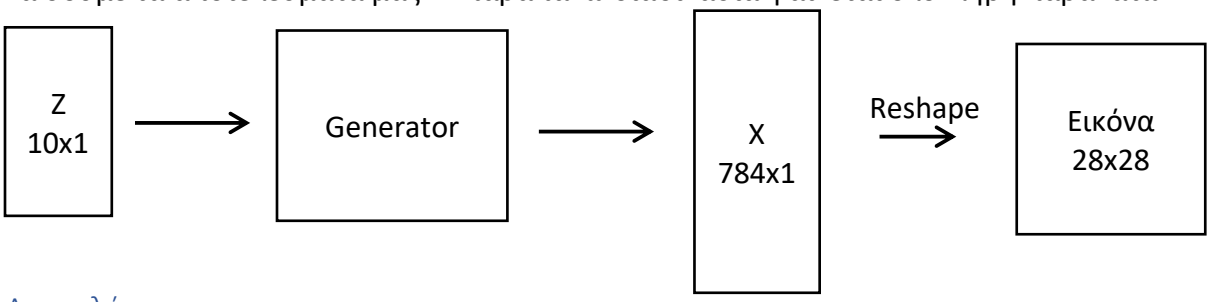
$$W_1 = A_1 * Z + B_1 \quad W_1: διάσταση\ 128\ x\ 1$$

$$Z_1 = \max\{W_1, 0\}\,(ReLU) \quad Z_1: διάσταση\ 128\ x\ 1$$

$$W_2 = A_2 *\ Z_1 + B_2 \quad W_2: διάσταση\ 784\ x\ 1$$

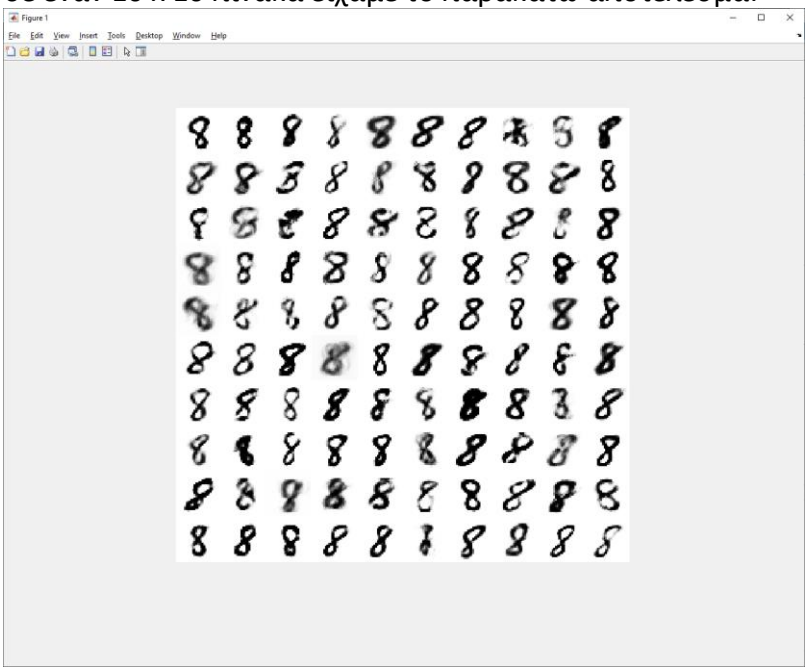
$$X = \frac{1.}{1 + \exp(W_2)}\,(Sigmoid) \quad X: διάσταση\ 784\ x\ 1$$

το οποίο μετατρέποντας το σε μια εικόνα 28x28, βάζοντας πρακτικά σε σειρά ανά 28 τα στοιχεία του, 28 φορές. Έτσι χρησιμοποιώντας την εντολή imshow(X) της MATLAB μπορούμε να δούμε τα αποτελέσματα μας. Η παραπάνω διαδικασία φαίνεται ολόκληρη παρακάτω:



Αποτελέσματα

Για εκατό υλοποιήσεις του Z που εφαρμόστηκαν στον generator πήραμε εκατό 28x28 εικόνες και βάζοντάς τες σε έναν 10 x 10 πίνακα είχαμε το παρακάτω αποτέλεσμα:



Πρόβλημα 2.2

Περιγραφή Προβλήματος

Για αυτό το πρόβλημα θέλουμε να κάνουμε inpainting με την βοήθεια του generator που είχαμε και στο πρόβλημα 2.1. Συγκεκριμένα θέλουμε η έξοδος του generator να είναι ένα συγκεκριμένο ιδανικό ''χειρόγραφο'' οκτώ όμως η σύγκρισή της δεν θα γίνεται με το ιδανικό οκτώ αλλά με ένα το οποίο είναι κομμένο, δηλαδή έχουμε ''πετάξει'' pixel της εικόνας και έχουμε προσθέσει και θόρυβο, αφού εφαρμόσουμε βέβαια τον ίδιο μετασχηματισμό T που εφαρμόσαμε στην ιδανική εικόνα και στην έξοδο του generator. Για να πάρουμε το προς επεξεργασία διάνυσμα X_n έχουμε εφαρμόσει τον παρακάτω μετασχηματισμό στην ιδανική εικόνα:

$$X_n = T\,X + \theta\acute{o}\rho\upsilon\beta o\varsigma$$

όπου το T είναι μία μήτρα Nx784 της μορφής [I 0], όπου I είναι ο μοναδιαίος πίνακας διάστασης N x N, το 0 είναι μηδενικός πίνακας διάστασης N x (784-N) και N ο αριθμός των στοιχείων που θέλουμε να κρατήσουμε από την αρχική εικόνα. Πρακτικά με αυτό το T κρατάμε N στοιχεία από την αρχική εικόνα X.

Έτσι έχοντας το προς επεξεργασία διάνυσμα X_n πρέπει να ελαχιστοποιήσουμε το κόστος J(Z) ως προς το Z. Το κόστος δίνεται από την παρακάτω εξίσωση:

$$J(Z) = N * \log(||X_n - TG(Z)||^2) + ||Z||^2$$

Άρα ψάχνουμε το διάνυσμα Z που άμα δώσουμε σαν είσοδο στον generator τότε η έξοδός του, στην οποία εφαρμόζουμε τον ίδιο μετασχηματισμό T που εφαρμόστηκε και στα ιδανικά οκτώ, είναι όσο πιο κοντά γίνεται στο X_n. Επιπλέον με την παραπάνω συνάρτηση κόστους λαμβάνουμε υπόψιν μας ότι το Z είναι gaussian, με τον όρο $||Z||^2$.

Για να πετύχουμε την ελαχιστοποίηση του κόστους J(Z) ως προς το Z θα χρησιμοποιούμε gradient descent. Χρειαζόμαστε λοιπόν το gradient του J ως προς το Z. Για να το βρούμε ακολουθούμε την παρακάτω διαδικασία:

$$Ορίζουμε\ αρχικά\colon \Phi(X) = \log(\|X_n - TX\|^2)$$

όπου X η έξοδος του generator. Έπειτα υπολογίζουμε τα παρακάτω:

$$u_2 = \nabla_x \Phi(X) = -\frac{2}{\left|\left|X_n - TX\right|\right|^2} * T^T * (X_n - TX)$$

$$V_2 = u_2 \odot f_2'(W_2)\ ,όπου\ f_2'\ η\ παράγωγος\ της\ Sigmoid$$

$$u_1 = A_2^T * V_2$$

$$V_1 = u_1 \odot f_1'(W_1)\ όπου\ f_1'\ η\ παράγωγος\ της\ ReLU$$

$$u_0 = A_1^T * V_1$$

$$όμως\ u_0 = \nabla_Z \Phi(X)$$

Άρα το gradient του κόστους J(Z) ως προς το Z προκύπτει από την παρακάτω σχέση:

$$\nabla_Z\{J(Z)\} = \nabla_Z\{N * \log(\|X_n - TX\|^2) + ||Z||^2\} = N * u_0 + 2 * Z$$

όπου όπως ειπώθηκε και παραπάνω N είναι το μήκος του προς επεξεργασία διανύσματος X_n , T ο μετασχηματισμός που εφαρμόσαμε στα ιδανικά οκτώ και X είναι η έξοδος του generator για το διάνυσμα Z διαστάσεων 10 x 1 με τυχαίες τιμές από την gaussian N(0,1).

Επιπλέον χρησιμοποιούμε και την μέθοδο ADAM με την οποία κανονικοποιούμε την παράγωγο ώστε να έχουμε πιο ομαλή σύγκλιση. Για να το πετύχουμε αυτό αρχικά βρίσκουμε την ενέργεια με τον παρακάτω τρόπο:

$$P_t = (1 - \lambda) * P_{t-1} + \lambda * \nabla_Z\{J(Z)\}_{t-1}$$

έχοντας αρχικά λ = 1 και μετά τον πρώτο υπολογισμό το κάνουμε μια πολύ μικρή τιμή, συγκεκριμένα λ = 0.001 όπως φαίνεται και στην ενότητα ΚΩΔΙΚΕΣ.

Τέλος εφαρμόζουμε τον gradient descent μαζί με την μέθοδο ADAM στην είσοδο Z του generator έτσι ώστε να ελαχιστοποιήσουμε το κόστος J(Z), όπως φαίνεται παρακάτω:

$$Z_t = Z_{t-1} - \mu * \frac{\nabla_Z\{J(Z)\}_{t-1}}{\sqrt{c + P_{t-1}}}$$

όπου μ ο ρυθμός μάθησης και c ένας πάρα πολύ μικρός αριθμός, συγκεκριμένα 10^-6 όπως φαίνεται και στην ενότητα ΚΩΔΙΚΕΣ.

Αποτελέσματα

Σε κάθε περίπτωση κρατάμε διαφορετικό αριθμό pixel από την προς επεξεργασία εικόνα και δοκιμάζουμε η ίδια είσοδος Z πόσο καλά μπορεί να προσεγγίσει το κάθε ένα από τα τέσσερα ιδανικά οκτώ.

Στις αριστερά εικόνες η κάθε στήλη είναι η προσπάθεια του ίδιου Z να κάνει inpainting στο οκτώ που έχει προστεθεί θόρυβος και έχουν αφαιρεθεί pixel έτσι ώστε να προσεγγίζει το ζητούμενο ιδανικό οκτώ που φαίνεται από πάνω του. Στην πρώτη σειρά βλέπουμε τα τέσσερα ιδανικά οκτώ, στην δεύτερη σειρά τα οκτώ με τον θόρυβο και τα ‘’χαμένα’’ pixel και τέλος στην τρίτη σειρά το τελικό αποτέλεσμα του generator.

Στην δεξιά εικόνα βλέπουμε την μείωση του κόστους σε κάθε περίπτωση.

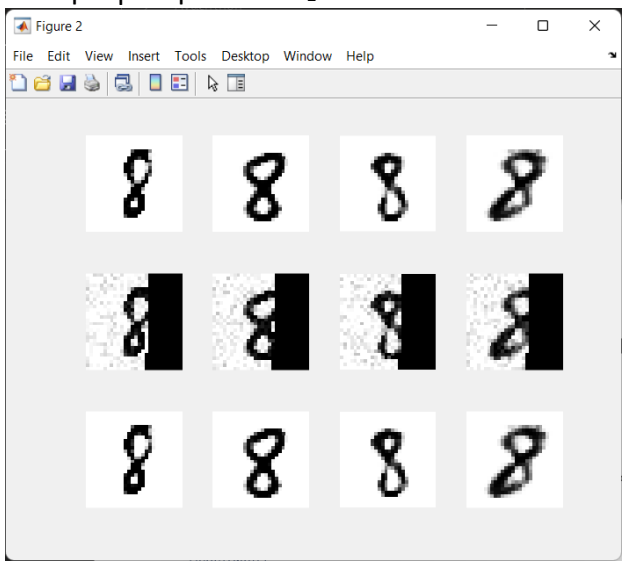
Παρακάτω φαίνονται οι τρείς περιπτώσεις που δοκιμάστηκαν, συγκεκριμένα στην $1^{η}$ περίπτωση κρατήθηκαν 500 pixel της προς επεξεργασία εικόνας στην δεύτερη 400 και τέλος στην τρίτη 300. Επιπλέον σε κάθε περίπτωση γίνεται inpainting των τεσσάρων οκτώ από τρείς διαφορετικές εισόδους Z.

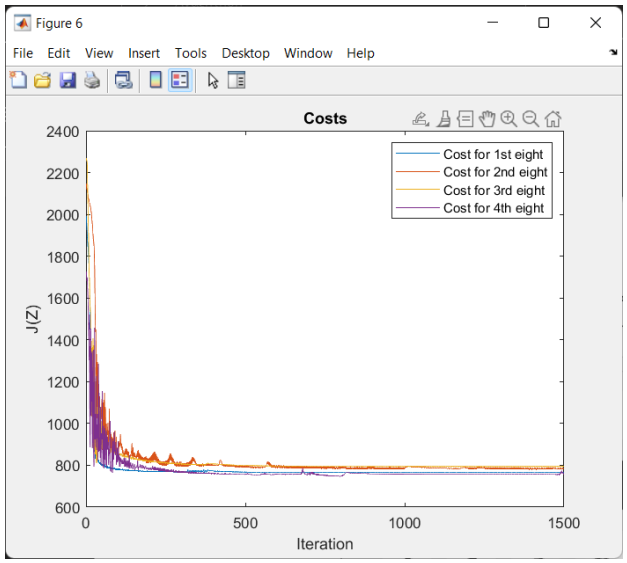
Τέλος και για τις τρείς διαφορετικές περιπτώσεις χρησιμοποιήθηκε learning rate = 0.05 για τον gradient descent αλγόριθμο και 1500 iterations.

Περίπτωση 1η

Αποτελέσματα για τρία διαφορετικά Z και N = 500, δηλαδή 284 ‘’χαμένα’’ pixel.

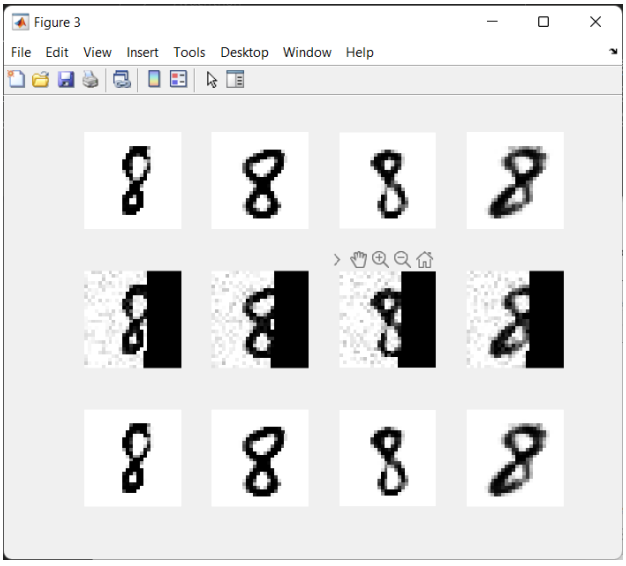
Για την πρώτη είσοδο Z_1:

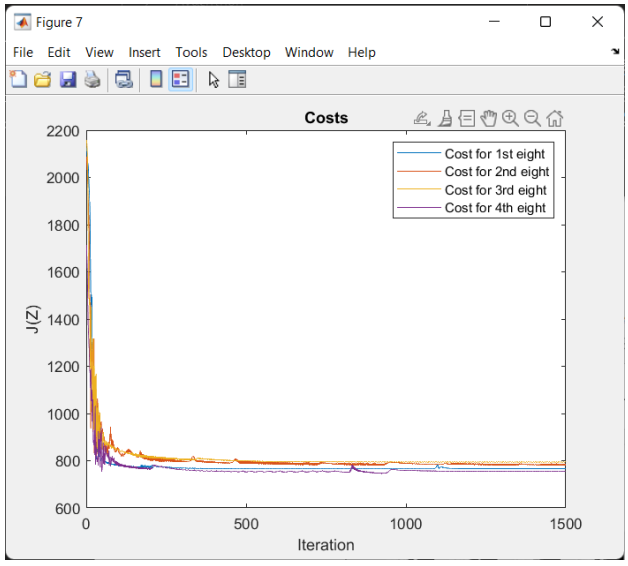




Όπως φαίνεται και από τις εικόνες αλλά και από την σύγκλιση των κοστών τα αποτελέσματα (εικόνες της 3ης σειράς) είναι πάρα πολύ καλα καθώς ο θόρυβος έχει αφεραιθεί πλήρος αλλά και τα κομμάτια που είχαν χαθεί έχουν ζωγραφιστεί σωστά και είναι πρακτικά ίδια με τις ιδανικές εικόνες

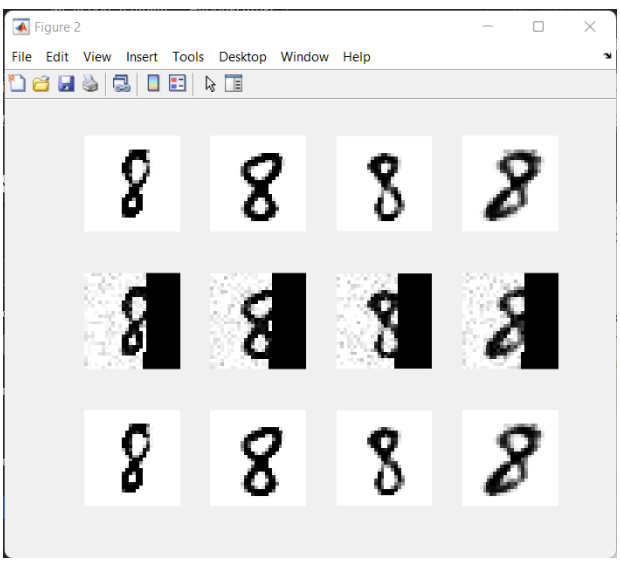
Για την δευτερη είσοδο Z_2:

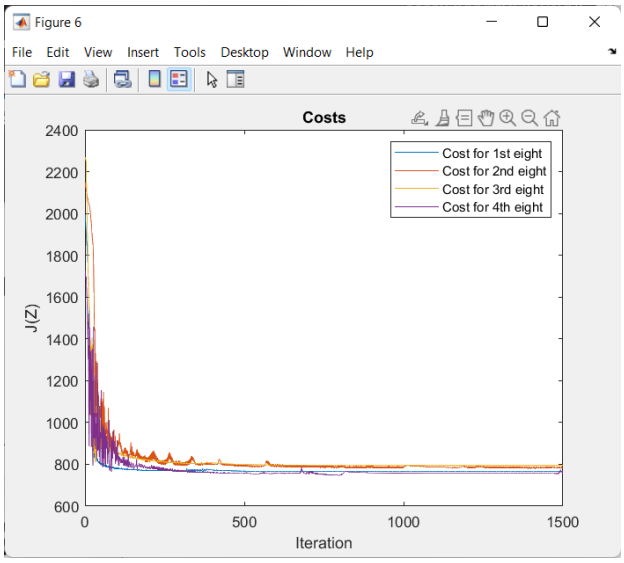




Και για την εισόδο Z_2 όπως και για την Z_1 τα αποτελέσματα είναι παρα πολύ καλα πράγμα το οποίο φαίνεται πάλι και οπτικά αλλά και από την σύγκλιση των κόστων.

Για την τρίτη είσοδο Z_3:

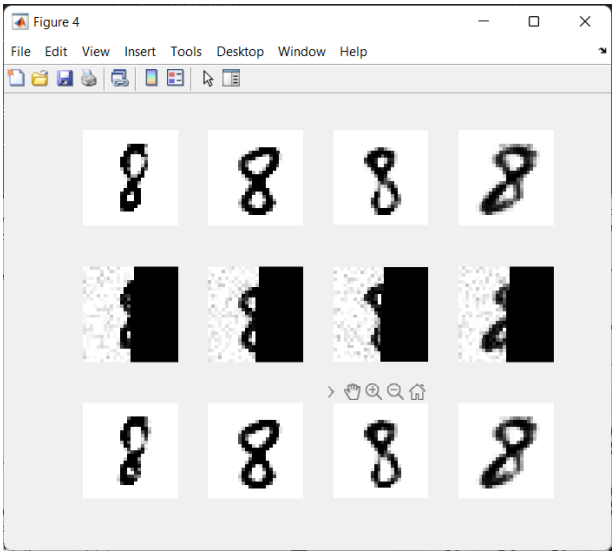


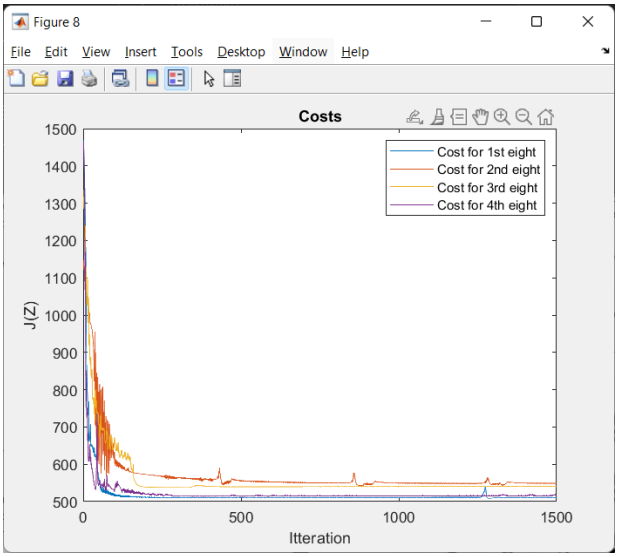


Τέλος και για την είσοδο Z_3 η τελική έξοδος του generator είναι πάρα πολύ καλή όπως φαίνεται πάλι και οπτικά αλλά και από την σύγκλιση των κοστών.

Περίπτωση 2η

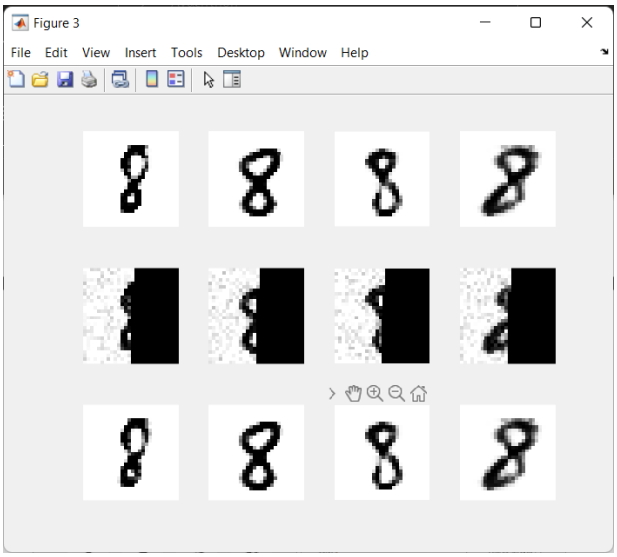
Αποτελέσματα για τρία διαφορετικά Z και N = 400, δηλαδή 384 ''χαμένα'' pixel.
Για την πρώτη είσοδο Z_1:

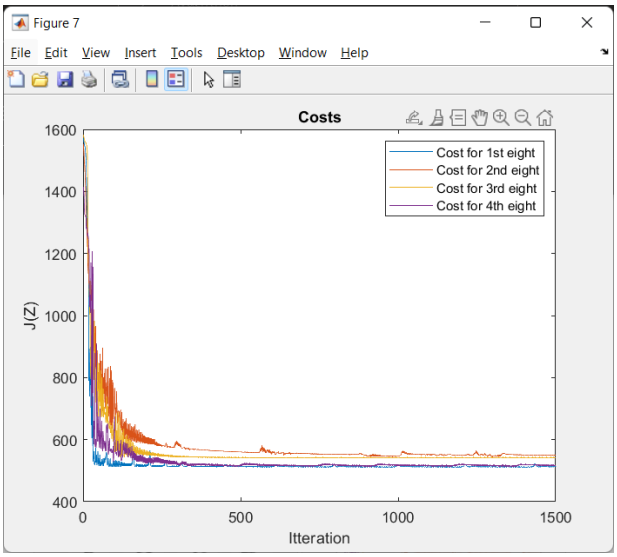




Η έξοδος του generator για το Z_1, $G(Z_1)$ όπως βλέπουμε παραπάνω, προσεγγίζει πάρα πολύ καλά και τα τέσσερα ιδανικά οκτώ το οποίο φαίνεται και στα κόστη τα οποία όλα συγκλίνουν. Το κόστος για το δεύτερο οκτώ φαίνεται να συγκλίνει λίγο πιο πάνω από τα υπόλοιπα αλλά και για αυτό το οκτώ έχουμε πολύ καλό αποτέλεσμα.

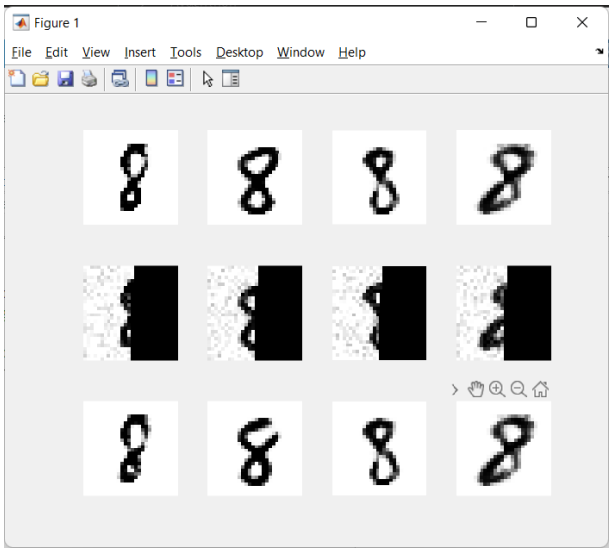
Για την δεύτερη είσοδο Z_2:

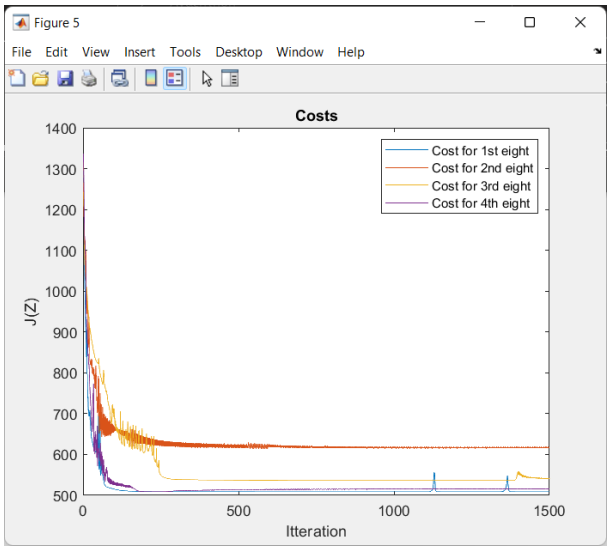




Και για το Z_2 τα αποτελέσματα είναι παρα πολύ καλά καθώς προσεγγίζει και τα τέσσερα ιδανικά οκτώ με το δελύτερο και το τρλιτο να συγκλίνουν σε λίγο μεγαλύτερο κόστος από τα άλλα δύο.

Για την τρίτη είσοδο Z_3:



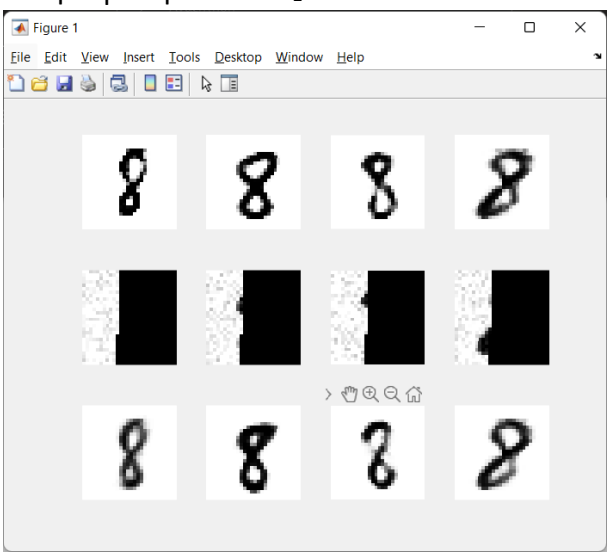


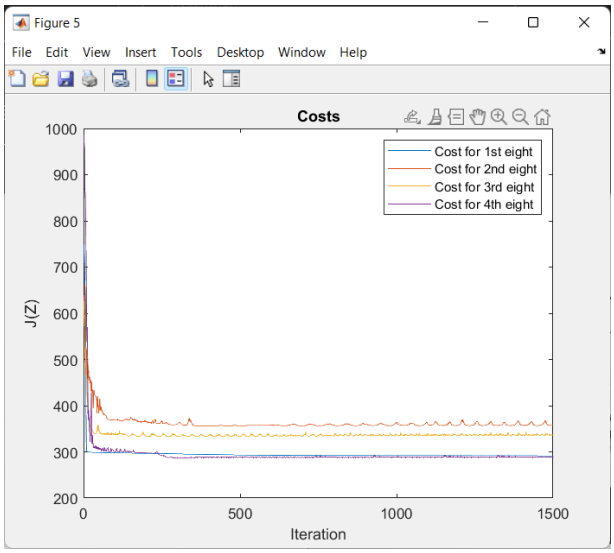
Στη συγκεκριμένη περίπτωση όπως βλέπουμε στην αριστερά εικόνα, αλλά και από την σύγκλιση των κοστών η έξοδος του generator για το Z_3, $G(Z_3)$ προσεγγίζει καλά το πρώτο, το τρίτο και το τέταρτο ιδανικό οκτώ αλλά για το δεύτερο δεν έχει τόσο καλά αποτελέσματα.

Και σε αυτή την περίπτωση έχουμε πολύ καλά αποτελέσματα παρά τα λιγότερα pixel που κρατάμε από τις προς επεξεργασία εικόνες σε σχέση με την πρώτη περίπτωση.

Περίπτωση 3η

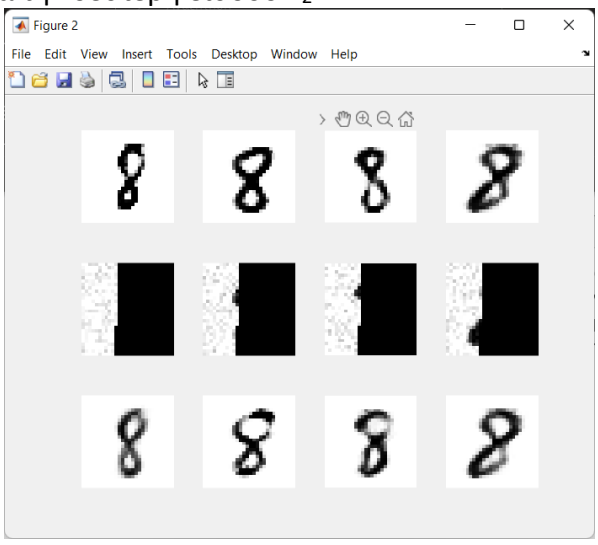
Αποτελέσματα για τρία διαφορετικά Z και N = 300, δηλαδή 484 ''χαμένα'' pixel.
Για την πρώτη είσοδο Z_1:

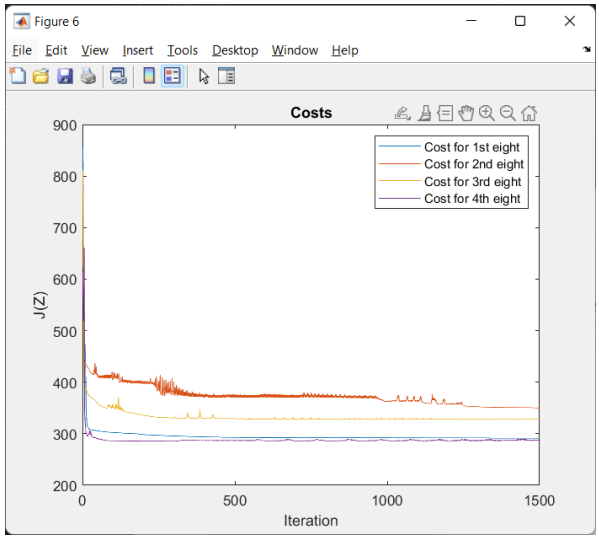




Όπως φαίνεται από τα αποτελέσματα πλεόν ο generator δυκολεύεται να ''ζωγραφίσε'' ακριβώς ίδια με τα ιδανικά καθώς όπως φαίνετια από τις εικόνες τις δεύτερης σειράς τα οκτώ δεν φαίνονται καθόλου καθώς χάνουμε πολύ΄μεγάλο αριθμό pixel. Ωστόσο και παλι τα αποτελέσματα είναι θεματικά ειδικά στο πρώτο και στο τέταρτο οκτώ τα οποία δεν είναι παρα πολύ ''μακρια'' από τα ιδανικά.

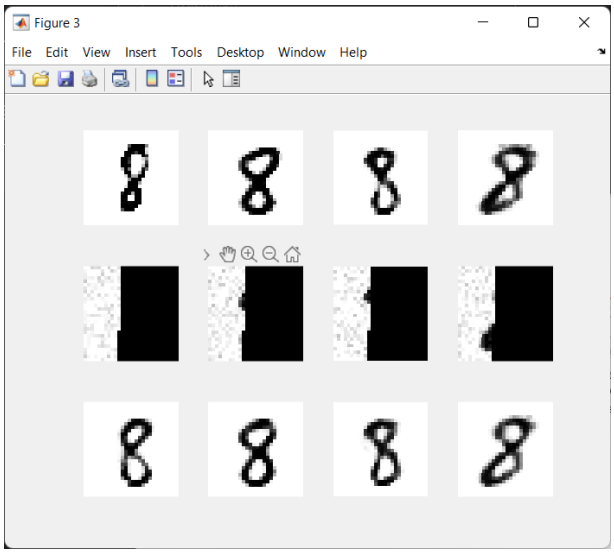
Για την δεύτερη είσοδο Z_2:

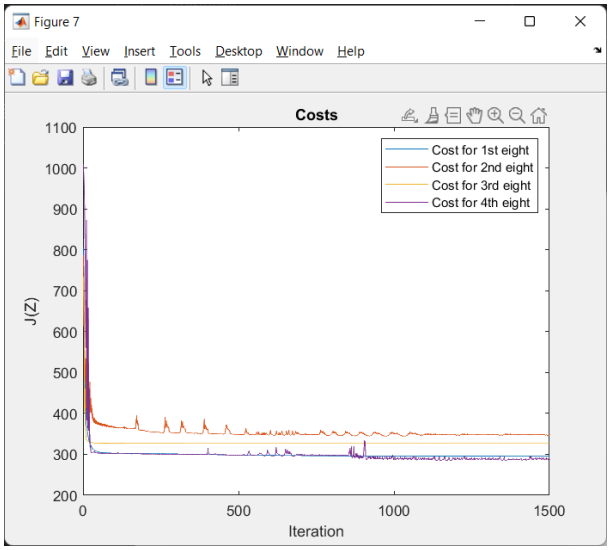




Ότι ίσχυε για την είσοδο Z_1 ισχύει και για την Z_2 καθώς πάλι ο generator δεν μπορεί να πετύχει ακριβώς τα ιδανικά οκτώ αλλά για το πρώτο και το τέταρτο το αποτέλεσμα είναι αρκετά κοντά σε αντίθεση με το δεύτερο και το τρίτο που για τα οποία γενικά φαίνεται να δυσκολεύεται περισσότερο να τα ''ζωγραφίσει''.

Για την τρίτη είσοδο Z_3:





Τέλος και για την τρίτη είσοδο Z_3 ο generator πάλι με τόσα λίγα pixel δεν μπορεί να προσεγγίσει τα ιδανικά οκτώ πέρα από το τέταρτο το οποίο είναι πιο κοντά στο ιδανικό από τα άλλα τρία.

Γενικά από τις διάφορες δοκιμές φάνηκε πως η διαδικασία έχει πάρα πολύ καλά αποτελέσματα για τιμές N μέχρι και 400~350 pixel και είναι πολύ εντυπωσιακό καθώς οριακά χωρίς να βλέπει το μισό οκτώ μπορεί να το αναπληρώσει.

Πρόβλημα 2.3

Περιγραφή Προβλήματος

Το τελευταίο πρόβλημα είναι παρόμοιο με το πρόβλημα 2.2 με μόνη διαφορά τον μετασχηματισμό T που εφαρμόζουμε στις ιδανικές εικόνες και στην έξοδο του generator. Συγκεκριμένα εδώ δεν έχουμε εικόνες με θόρυβο και χαμένα pixel αλλά αυτό που κάνουμε είναι να παίρνουμε τις εικόνες 28 x 28 και αφού τις μετατρέψουμε σε διανύσματα 784 x 1 τους εφαρμόζουμε τον μετασχηματισμό T ο οποίος τις κάνει διανύσματα 49 x1 δηλαδή εικόνες 7 x 7. Για να το πετύχουμε αυτό χρησιμοποιούμε τον μετασχηματισμό T που περιγράφεται παρακάτω.

Πρακτικά χρησιμοποιούμε έναν πίνακα διάστασης 49 x 784 του οποίου η κάθε γραμμή όταν πολλαπλασιαστεί με το διάνυσμα X διάστασης 784 x 1, δηλαδή την αρχική εικόνα 28 x 28 υψηλής ανάλυσης μας δίνεις τις τιμές των pixel μίας εικόνα χαμηλής ανάλυσης 7 x 7, δηλαδή παίρνουμε ένα διάνυσμα 49 x 1 το οποίο περιέχει αυτές τις τιμές. Για να το πετύχουμε αυτό παίρνουμε τον μέσο όρο από 4 x 4 κουτάκια τις αρχικής εικόνας βάζοντας στον T την τιμή 1/16 σε συγκεκριμένες θέσεις τις κάθε γραμμής του και στις υπόλοιπες μηδενικά. Οι θέσεις που με τιμή 1/16 φαίνονται παρακάτω

$$1\eta\ \gamma\rho\alpha\mu\mu\acute{\eta}: i * 28 + (0 * 4 : 1 * 4 - 1)$$

$$2\eta\ \gamma\rho\alpha\mu\mu\acute{\eta}: i * 28 + (1 * 4 : 2 * 4 - 1)$$

$$\vdots$$

$$7\eta\ \gamma\rho\alpha\mu\mu\acute{\eta}: i * 28 + (6 * 4 : 2 * 4 - 1)$$

$$\vdots$$

$$\gamma\iota\alpha\ i = 0, 1, 2, 3$$

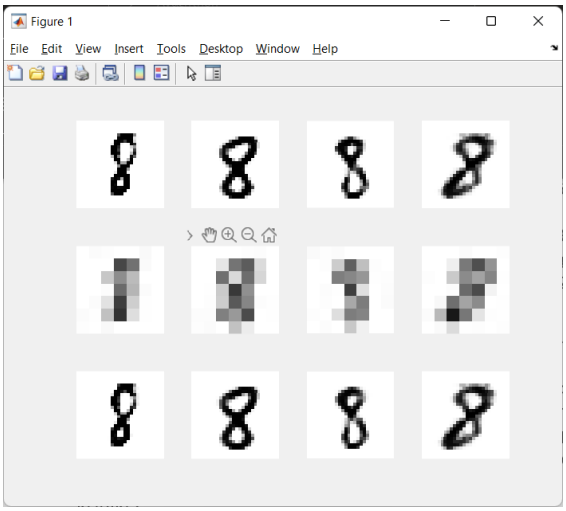
Πρακτικά έτσι καταλήγουμε να έχουμε έναν πίνακα T 49 x 784 στον στις γραμμές πηγαίνουμε και βάζουμε τέσσερεις τετράδες με την τιμή 1/16 οι οποίες απέχουν μεταξύ τους 28 κελιά στα οποία βάζουμε μηδενικά. Οι τετράδες αυτές δεν μπαίνουν στις ίδιες στήλες σε κάθε γραμμή αλλά κάθε γραμμή που κατεβαίνουμε κάνουμε ένα shift δεξιά. Μετά από την $7^{η}$ γραμμή θα πάμε στην $8^{η}$ στην οποία ξεκινάμε να βάζουμε τετράδες 1/16 μετά την στήλη που έχει μπει το τελευταίο 1/16 στην $7^{η}$ γραμμή και μέχρι και εκεί έχουμε βάλει μόνο μηδενικά. Αυτές οι τετράδες πάλι θα απέχουν μεταξύ τους 28 κελιά και κάθε φορά που θα κατεβαίνουμε γραμμή θα κάνουμε ένα shift δεξιά. Πάλι όταν βάλουμε όλες τις τετράδες στις 7 γραμμές σταματάμε και συνεχίζουμε για τις επόμενες 7 και συνεχίζουμε παρομοίως μέχρι να έχουμε τον τελικό πίνακα T.

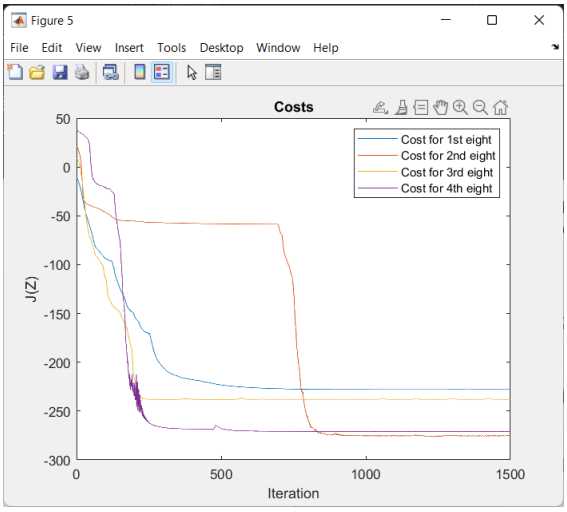
Αφού τώρα έχουμε τον transformation matrix που θέλουμε η διαδικασία που ακολουθούμε είναι ίδια με αυτήν του προβλήματος 2.2 δηλαδή βρίσκουμε το gradient της J(Z) ως προς το Z με τον τρόπο που ειπώθηκε παραπάνω, εφαρμόζουμε την μέθοδο ADAM για την κανονικοποίηση των παραγώγων και τέλος εφαρμόζουμε τον gradient descent αλγόριθμο στην είσοδο του generator Z έτσι ώστε να ελαχιστοποιήσουμε το J(Z).

Αποτελέσματα

Παρακάτω φαίνονται τρεις δοκιμές ανακατασκευής των ιδανικών εικόνων από ένα Z κάθε φορά. Σε όλες τις περιπτώσεις χρησιμοποιήθηκε learning rate = 0.005 για τον gradient descent αλγόριθμο και 1500 iterations. Επίσης όπως και στο πρόβλημα 2.2 στις αριστερά εικόνες φαίνονται οι ιδανικές 28 x 28 εικόνες στην πρώτη σειρά και η τελευταία έξοδος του generator στην τρίτη σειρά ενώ στην δεύτερη σειρά φαίνονται η μεγεθυμένες 7 x 7 σε 28 x 28 προς επεξεργασία στις οποίες η μεγέθυνση είναι μόνο για την εμφάνιση τους στα αποτελέσματα και έχει γίνει με την εντολή kron() της MATLAB το οποίο φαίνεται και πιο αναλυτικά στην ενότητα ΚΩΔΙΚΕΣ.

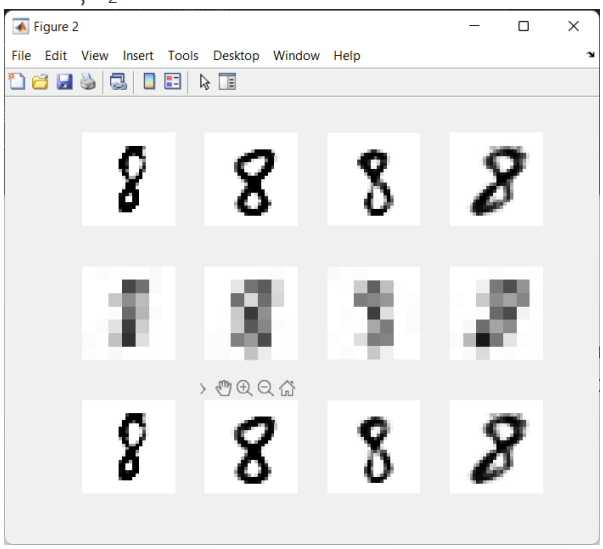
Είσοδος Z_1

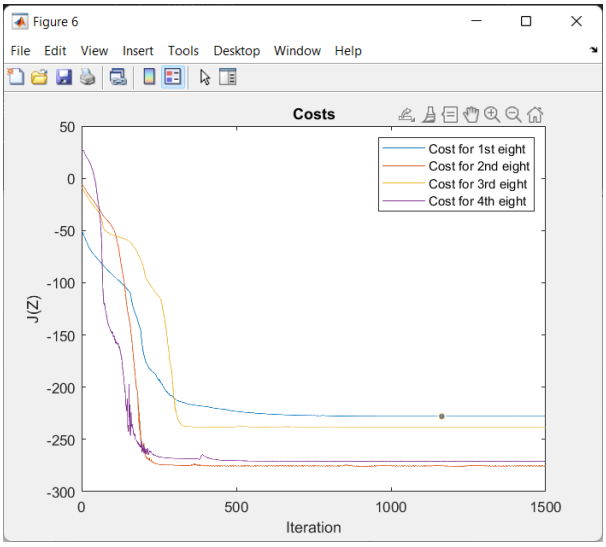




Τα αποτελέσματα όπως φαίνεται και οτικά αλλά και από την σύγκλιση είναι εκπηκτικά και αν και για το 2^{ο} ιδανικό οκτώ παίρνει λιγο περισσοτερα βήματα στον αλγόριθμο έτσι ώστε να συγκλίνει και τα τέσσερα οκτώ της 3^{ης} σειράς προσεγγίζουν πάρα πολύ καλά τα τέσσερα ιδανικά για το Z_1.

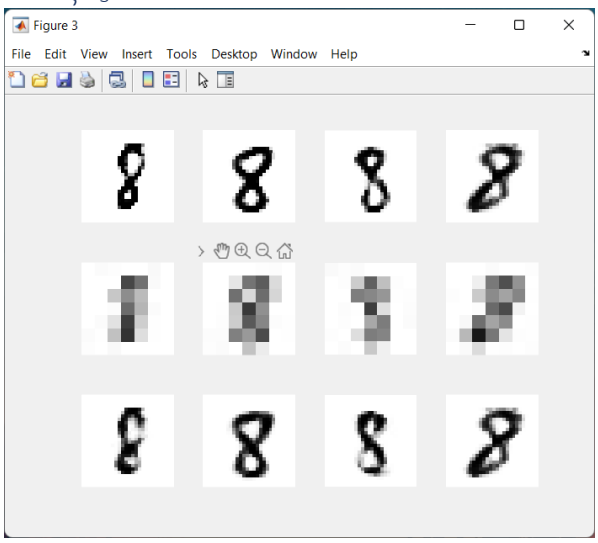
Είσοδος Z_2

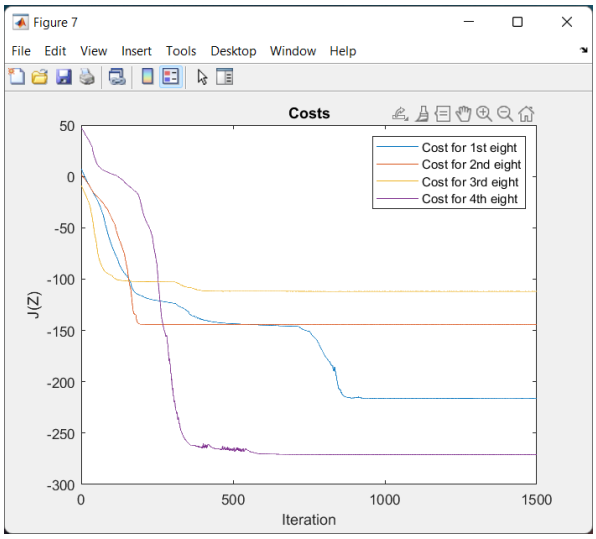




Για την είσοδο Z_2 πάλι τα αποτελέσματα είναι πάρα πολύ καλά για κάθε ιδανικό οκτώ.

Είσοδος Z_3





Για την είσοδο Z_3 όπως φαίνεται και από τις εικόνες αλλά και από την σύγκλιση των κοστών δεν γίνεται τόσο κάλη προσέγγιση των πρώτον τριών οκτώ ωστόσο για το τέταρτο τα αποτελέσματα είναι πάλι πολύ καλά.

Γενικά με τις διάφορες δοκιμές για τον ρυθμό μάθησης 0.005 πάρθηκαν τα καλύτερα αποτελέσματα με τον generator να καταλήγει σε οκτάρια τα οποία ητάν ακριβή σε κάθε λεπτομέρια με τα αρχικά όπως φάνηκε και στα τρία παραπάνω παραδείγματα.

Κώδικες

Κώδικας Προβλήματος 2.1

```
%% Task 2.1
clc;
clear;
close all;
%% Import the data of the generator
generator = load('data21.mat');
A1 = generator.A_1;
A2 = generator.A_2;
B1 = generator.B_1;
B2 = generator.B_2;
%% Create inputs Z ~ N(0,1)
sz = [10, 1, 100];
inputs = randn(sz);  % Create the generator inputs
%% Calculate the Generator Output
output = zeros(784,1,100);
for i=1:100
    Z = inputs(:,:,i); % Take a single input

    % First Layer
    W1 = A1 * Z + B1;  % Pass through the first layer
    Z1 = ReLU(W1);  % Pass through the Activation Function (ReLU)
    % Second Layer
    W2 = A2 * Z1 + B2;  % Pass through the second layer
    X = sigmoid(W2);  % Pass through the Activation Function (Sigmoid)

    output(:,:,i) = X; % Keep all the output vectors
end

%% Reshape the output to a picture
canvas = []; % Initialize the 10x10 picture "canvas"
row = []; % Initialize a temp row matrix for the "canvas"
cnt = 0; % Initialize a counter so we go to the next row every 10 pictures
for i=1:100
    X_2D = reshape(output(:,:,i), 28, 28); % Make the output a 28x28 matrix
    row = [row, X_2D]; % Add that matrix to the current row
    cnt = cnt + 1; % Increase the counter
    if cnt == 10
        canvas = [canvas; row]; % Add the current row to the canvas
        row = []; % Empty the row for the next pictures
        cnt = 0; % Set the counter to 0
    end
end
%% Show the image
imshow(canvas)  % Display the 100 eights

%% Activation Functions
function Z1 = ReLU(W) % Activation ReLU
    Z1 = max(W,0);
end
function X = sigmoid(W) % Activation Sigmoid
    X = 1./(1 + exp(W));
end
```

Κώδικας Προβλήματος 2.2

```
%% Task 2.2
clc;
clear;
close all;

%% Load the Generator
generator = load('data21.mat');
A1 = generator.A_1;
A2 = generator.A_2;
B1 = generator.B_1;
B2 = generator.B_2;


%% Load the Matricies
data = load("data22.mat");
X_ideal = data.X_i;  % The ideal vectors *NOT FOR PROCESSING*
X_processing = data.X_n;  % The vectors with added noise


%% Choose the data and initialize the transformation matrix
N = 400;  % The amount of data we keep

I = eye(N);  %Identity Matrix
Zero = zeros([N, 784-N]);  % Zero Matrix
T = [I Zero];  % Transform Matrix


X_ideal = reshape(X_ideal,28,28,4);


Xn = T*X_processing;  % Apply the transformation T to the processing images
Xn =  reshape(Xn,N,1,4);


Xn_paint = Xn;
Xn_paint(N:784,:,:) = 0;  % Add zeros to the images to paint them as 28x28
Xn_paint = reshape(Xn_paint,28,28,4);


%% Calculate Cost
inputs_size = [10, 1, 10];
costs_size = [1500,4,4];
generator_inputs = randn(inputs_size);  % Initialize the generator inputs
costs = zeros(costs_size);  % Initialize a cost matrix

lr = 0.05;  % Learning rate for gradient descent
```

```
for i=1:4 % The amount Z's that we try as generator inputs
    Z = generator_inputs(:,:,i);  % Generator Input
    figure(i)
    for j=1:4
        Z_tmp = Z;  % Try the same Z for every target each time
        Xn_j = Xn(:,:,j);  % The generator Targets
        % Initialize Adam Parameters
        lambda = 1;
        power = 0;
        c = 10^-6;
        for itter=1:1500
            % First Layer
            W1 = A1 * Z_tmp + B1;  % Pass though the first layer
            Z1 = ReLU(W1);  % Pass through the Activation Function (ReLU)

            % Second Layer
            W2 = A2 * Z1 + B2;  % Pass through the second layer
            X = sigmoid(W2);  % Pass through the Activation Function (Sigmoid)

            J = cost(Z_tmp,Xn_j,X,N,T);  % Calculate the cost
            costs(itter,j,i) = J;

            % Calculate the cost J(Z) gradient
            u2 = derivativePhi(Xn_j,X,T);
            v2 = u2.* DerivativeSigmoid(W2);
            u1 = A2' * v2;
            v1 = u1.* DerivativeReLU(W1);
            u0 = A1' * v1;
            grad = N * u0 + 2 * Z_tmp;

            % Adam normalization
            power = (1-lambda) * power + lambda * grad.^2;
            lambda = 0.001; % Change lambda to a small value

            % Gradient Descent Algorithm
            Z_new = Z_tmp - lr * grad./sqrt(power +  c);
            Z_tmp = Z_new;
        end
        subplot(3,4,j)
        imshow(X_ideal(:,:,j))
        subplot(3,4,j+4)
        imshow(Xn_paint(:,:,j))
        subplot(3,4,j+8)
        imshow(reshape(X,28,28))
    end
end
%% Plot the costs
for i=1:4
    figure()
    plot(costs(:,:,i))
    ylabel('J(Z)')
    xlabel('Iteration')
    title('Costs')
    legend('Cost for 1st eight', 'Cost for 2nd eight', 'Cost for 3rd eight', ...
           'Cost for 4th eight')
end
```

```
%% Functions
function J = cost(Z,Xn,X,N,T) % Cost Function
    J = N * log(norm((Xn - T * X))^2) + norm(Z)^2;
end


function u2 = derivativePhi(Xn,X,T) % Phi Derivative
    u2 = (-2/(norm(Xn - T*X)^2)) * T' * (Xn - T*X);
end


function Z1 = ReLU(W)  % Activation ReLU
    Z1 = max(W,0);
end


function f1_der = DerivativeReLU(W)  % ReLU Derivative
    W(W(:,:)<0) = 0;
    W(W(:,:)>0) = 1;
    f1_der = W;
end


function X = sigmoid(W)  % Activation Sigmoid
    X = 1./(1 + exp(W));
end


function f2_der = DerivativeSigmoid(W)  % Sigmoid Derivative
    f2_der = -exp(W)./((1 + exp(W)).^2);
end
```

Κώδικας Προβλήματος 2.3

```
%% Task 2.3
clc;
clear;
close all;

%% Load the Generator
generator = load('data21.mat');
A1 = generator.A_1;
A2 = generator.A_2;
B1 = generator.B_1;
B2 = generator.B_2;

%% Load the Matricies
data = load("data23.mat");
X_ideal = data.X_i;  % The ideal vectors *NOT FOR PROCESSING*
X_processing = data.X_n;  % The vectors with added noise
```

```
%% Choose the data and initialize the transformation matrix
N = 49;
T_size = [N, 784];

mean = 1/16;
T = zeros(T_size);
cnt1 = 0;
cnt2 = 0;
for i=0:48   % Algorithm to make the transformation matrix T
    if mod(i,7) == 0 % Change every seven lines to place to the correct cells
        cnt1 = 0;
        cnt2 = 4 * 28 * int16(i/7);
    end
    for j = 0*28 : 28 : 3*28
        T(i+1, (j+1)+cnt2+cnt1:(j+4)+cnt2+cnt1) = mean; % Set the quads
    end
    cnt1 = cnt1 + 4;
end

X_ideal = reshape(X_ideal,28,28,4); % The ideal vectors *NOT FOR PROCESSING*

Xn =  reshape(X_processing,N,1,4); % The 49 x 1 vectors for processing

Xn_paint = Xn;
Xn_paint = reshape(Xn_paint,7,7,4); % Enlarge these vectors to display them

%% Calculate Cost
inputs_size = [10, 1, 10];
costs_size = [1500,4,4];
generator_inputs = randn(inputs_size);  % Initialize the generator inputs
costs = zeros(costs_size);  % Initialize a cost matrix

lr = 0.005;  % Learning rate fro gradient descent

for i=1:4 % The amount Z's that we try as generator inputs
    Z = generator_inputs(:,:,i);  % Generator Input
    figure(i)
    for j=1:4
        Z_tmp = Z;  % Try the same Z for every target each time
        Xn_j = Xn(:,:,j);  % The generator Targets

        % Initialize Adam Parameters
        lambda = 1;
        power = 0;
        c = 10^-6;
        for itter=1:1500
            % First Layer
            W1 = A1 * Z_tmp + B1;  % Pass through the first layer
            Z1 = ReLU(W1);  % Pass through the Activation Function (ReLU)
            % Second Layer
            W2 = A2 * Z1 + B2;  % Pass through the second layer
            X = sigmoid(W2);  % Pass through the Activation Function (Sigmoid)
```

```
            J = cost(Z_tmp,Xn_j,X,N,T); % Calculate the cost
            costs(itter,j,i) = J;

            % Calculate the cost J(Z) gradient
            u2 = derivativePhi(Xn_j,X,T);
            v2 = u2.* DerivativeSigmoid(W2);
            u1 = A2' * v2;
            v1 = u1.* DerivativeReLU(W1);
            u0 = A1' * v1;
            grad = N * u0 + 2 * Z_tmp;

            % Adam normalization
            power = (1-lambda) * power + lambda * grad.^2;
            lambda = 0.001; % Change lambda to a small value

            % Gradient Descent Algorithm
            Z_new = Z_tmp - lr * grad./sqrt(power +  c);
            Z_tmp = Z_new;
        end
        subplot(3,4,j)
        imshow(X_ideal(:,:,j))
        subplot(3,4,j+4)
        imshow(kron(Xn_paint(:,:,j),ones(4)))
        subplot(3,4,j+8)
        imshow(reshape(X,28,28))
    end
end
%% Plot the costs
for i=1:4
    figure()
    plot(costs(:,:,i))
    ylabel('J(Z)')
    xlabel('Iteration')
    title('Costs')
    legend('Cost for 1st eight', ...
           'Cost for 2nd eight', ...
           'Cost for 3rd eight', ...
           'Cost for 4th eight')
end

%% Functions
function J = cost(Z,Xn,X,N,T) % Cost Function
    J = N * log(norm((Xn - T * X))^2) + norm(Z)^2;
end

function u2 = derivativePhi(Xn,X,T) % Phi Derivative
    u2 = (-2/(norm(Xn - T*X)^2)) * T' * (Xn - T*X);
end


function Z1 = ReLU(W)  % Activation ReLU
    Z1 = max(W,0);
end
```

```
function f1_der = DerivativeReLU(W)  % ReLU Derivative
    W(W(:,:)<=0) = 0;
    W(W(:,:)>0) = 1;
    f1_der = W;
end


function X = sigmoid(W)  % Activation Sigmoid
    X = 1./(1 + exp(W));
end


function f2_der = DerivativeSigmoid(W)  % Sigmoid Derivative
    f2_der = -exp(W)./((1 + exp(W)).^2);
end
```