ΜΗΧΑΝΙΚΟΙ Η/Υ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάκτηση εικόνας από βάση δεδομένων

Ψηφιακή Επεξεργασία και Ανάλυση Εικόνας

Κωνσταντοπούλου Ευαγγελία

1059560

1 Σύστημα ανάκτησης **PCA**

Η ανάλυση βασικών συστατικών (PCA) είναι ένα τυπικό εργαλείο στη σύγχρονη ανάλυση δεδομένων επειδή είναι μια απλή, μη παραμετρική μέθοδος για την εξαγωγή σχετικών πληροφοριών από σύνολα δεδομένων. Με ελάχιστη προσπάθεια, το PCA παρέχει έναν χάρτη πορείας για τον τρόπο μείωσης ενός σύνθετου συνόλου δεδομένων σε χαμηλότερη διάσταση για την αποκάλυψη μερικές φορές κρυμμένων, απλουστευμένων δομών που συχνά το υποστηρίζουν.

Υλοποίηση:

Αφότου πρώτα μετατρέψουμε τις εικόνες του training set σε grayscale, τις αποθηκεύουμε σε ένα μητρώο 100 x 1000. Υπολογίζουμε τον μέσο όλων των εικόνων και τον αφαιρούμε από κάθε εικόνα στο TrainSet. Αυτό έχει ως αποτέλεσμα το data set μας να έχει μέση τιμή μηδέν.

Υστερα υπολογίζουμε το Covariance μητρώο, πολλαπλασιάζοντας το μητρώο TrainSet με τον συζυγή μιγαδικό ανάστροφό του και διαιρώντας το με το πλήθος των εικόνων στο training set. Υπολογίζουμε τα ιδιοδιανύσματα του Covariance μητρώου και τα πολλαπλασιάζουμε από αριστερά με το μιγαδικό ανάστροφό του TrainSet. Το αποτέλεσμα το αποθηκεύουμε στο μητρώο transEigen το οποίο περιέχει τα τελικά ιδιοδιανύσματα μετασχηματισμένα στον αρχικό χώρο.

Έστω top οι μεγαλύτερες ιδιοτιμές που επιλέγουμε να κρατήσουμε. Τις αποθηκεύουμε σε έναν πίνακα N αφότου πρώτα τις κανονικοποιήσουμε για να σχηματίσουμε έναν ορθοκανονικό χώρο. Μόλις επιλέξουμε τα στοιχεία (eigenvectors) που θέλουμε να διατηρήσουμε στα δεδομένα μας, απλά παίρνουμε το ανάστροφο διάνυσμα και το πολλαπλασιάζουμε από τα αριστερά του αρχικού συνόλου δεδομένων, ανεστραμμένου. Θα μας δώσει τα αρχικά δεδομένα αποκλειστικά ως προς τα διανύσματα που αποφασίσαμε να κρατήσουμε, μειώνοντας έτσι τις διαστάσεις του αρχικού χώρου. Το αποτέλεσμα βρίσκεται στο TrainVect.

Αποθηκεύουμε τις εικόνες για testing σε ένα μητρώο 11 x 1000, TestSet. Αφαιρούμε από τις εικόνες στον συνολικό μέσο όπως και προηγουμένως, και ύστερα πολλαπλασιάζουμε πάλι τον ανεστραμμένο πίνακα N από τα αριστερά του ανεστραμμένου αρχικού συνόλου δεδομένων ώστε να βρεθούμε στον μικρότερο χώρο. Το αποτέλεσμα βρίσκεται στο TestVect.

Για κάθε στήλη του TrainVect υπολογίζουμε την απόσταση με κάθε στήλη του TestVect. Αποθηκεύουμε όλα τα αποτελέσματα στο διάνυσμα dist και βρίσκουμε σε ποιο δείκτη του υπάρχει η μικρότερη τιμή. Τους δείκτες αυτούς τους αποθηκεύουμε σε ένα διάνυσμα Tester. Μέσω αυτού βρίσκουμε το accuracy, δηλαδή πόσες εικόνες του test set ανακτήθηκαν επιτυχώς.

Για 10 μεταβλητές έχουμε accuracy 1, ενώ για 50 και 100 μεταβλητές έχουμε accuracy 3. Όπως βλέπουμε από τα αποτελέσματα, η μείωση του αριθμού των στοιχείων κοστίζει κάποια ακρίβεια και από την άλλη πλευρά, καθιστά το μεγάλο σύνολο δεδομένων απλούστερο, εύκολο στην εξερεύνηση και την απεικόνιση. Επίσης, μειώνει την υπολογιστική πολυπλοκότητα του μοντέλου που κάνει τους αλγόριθμους μηχανικής μάθησης να τρέχουν γρηγορότερα.

2 Σύστημα ανάκτησης **Autoencoders**

Οι Autoencoders είναι νευρωνικά δίκτυα που μπορούν να χρησιμοποιηθούν για τη μείωση των δεδομένων σε latent space και χαμηλή διάσταση, στοιβάζοντας πολλαπλούς μη γραμμικούς μετασχηματισμούς (επίπεδα). Έχουν αρχιτεκτονική κωδικοποιητή-αποκωδικοποιητή. Ο κωδικοποιητής χαρτογραφεί latent space και ο αποκωδικοποιητής ανακατασκευάζει την είσοδο. Εκπαιδεύονται με χρήση back propagation για ακριβή ανακα-

τασκευή της εισόδου. Κάνοντας τον latent space να έχει χαμηλότερες διαστάσεις από την είσοδο, μπορώ να χρησιμοποιήσω Autoencoders για μείωση της διάστασης.

Υλοποίηση:

Αφότου πρώτα μετατρέψουμε τις εικόνες του training set σε grayscale, τις αποθηκεύουμε σε ένα μητρώο 100 x 1000. Εκπαιδεύουμε τον Autoencoder μέσω αντίστοιχης συνάρτησης, επιλέγοντας κάθε φορά το hiddenSize, δηλαδή το μέγεθος της εξόδου του layer encoder. Μπορούμε να διαλέξουμε ανάμεσα στις 'logsig'(default) και 'satlin' ως συναρτήσεις μεταφοράς για τον encoder.

Logistic sigmoid function 'logsig':

$f(z) = \frac{1}{1+e^{-z}}$

Positive saturating linear transfer function 'satlin':

$$f(z) = \begin{cases} 0, & z \leq 0 \\ z, & 0 < z < 1 \\ 1, & z \geq 1 \end{cases}$$

Για τον layer decoder έχουμε τις ίδιες επιλογές και επιπλέον την 'purelin'.

Linear transfer function 'purelin':

$f(z) = z$

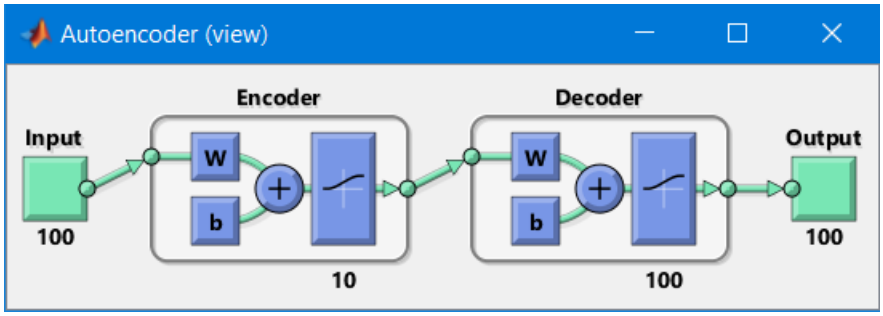
Τα 'MaxEpochs' (1000 by default) αποτελούν τις επαναλήψεις κατά το training του encoder. Όσες περισσότερες οι επαναλήψεις τόση ώρα παίρνει η εκπαίδευση του Autoencoder αλλά ταυτόχρονα μειώνει το μέσο τετραγωνικό σφάλμα. Το 'L2WeightRegularization' (0.001 by default) αποτελεί τον λ συντελεστή για τον L2 weight regularizer στη συνάρτηση (LossFunction). Η 'SparsityRegularization' αποτελεί τον μ συντελεστή της Loss Function, που ρυθμίζει την επίδραση του sparsity regularizer στη συνάρτηση κόστους. Το 'SparsityProportion' (0.05 by default) εκφράζει το επιθυμητό ποσοστό των training examples στα οποία θέλω να αντιδράσει το νευρωνικό. Αφότου πειραματίστηκα με τις τιμές των παραμέτρων αυτών κατέληξα σε κάποιες που προσφέρουν ικανοποιητικό handover χρόνου εκτέλεσης και τετραγωνικού σφάλματος.

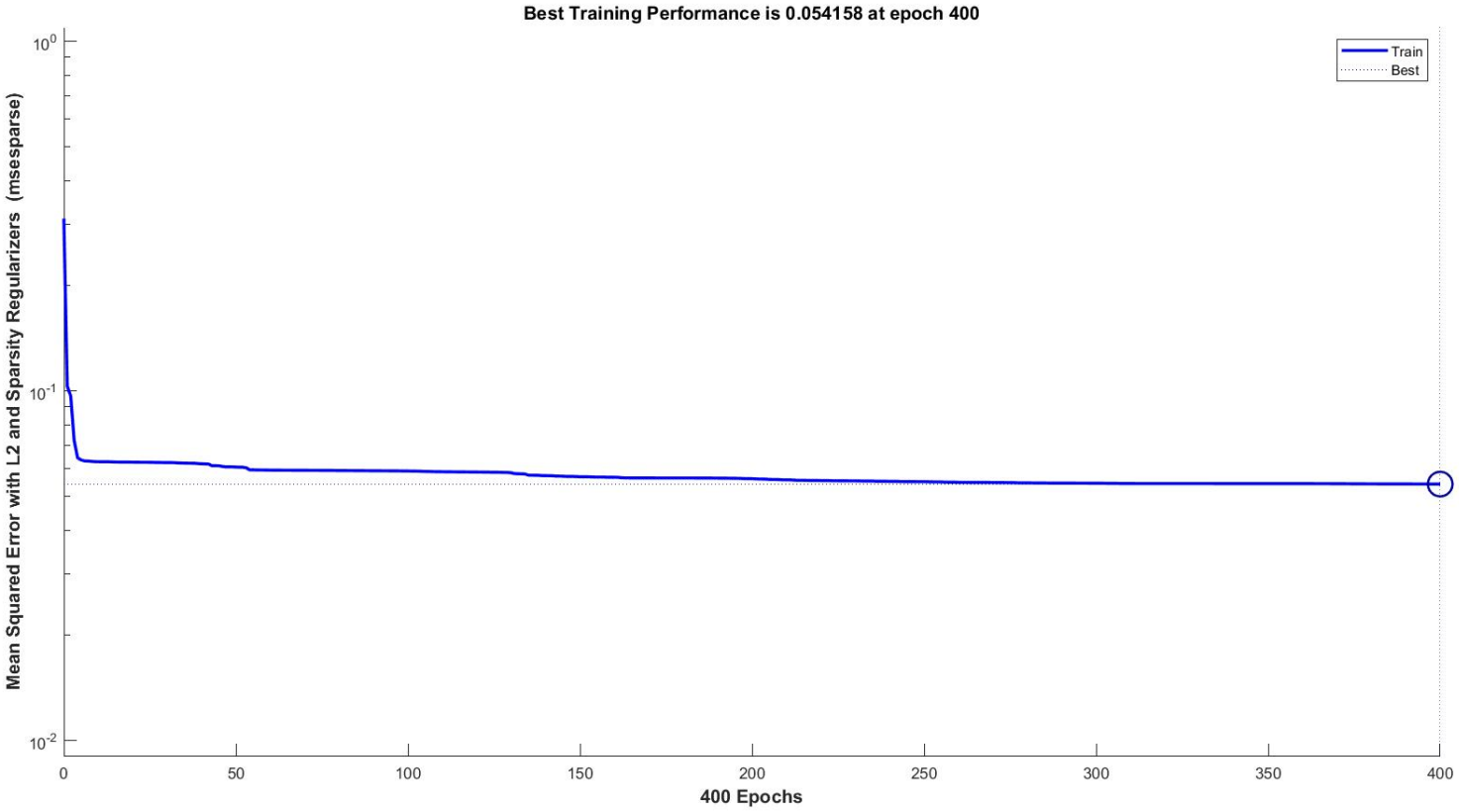
$$E = \underbrace{\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}\left(x_{kn} - \widehat{x}_{kn}\right)^2}_{\text{mean squared error}} + \lambda * \underbrace{\Omega_{weights}}_{L_2 \text{ regularization}} + \beta * \underbrace{\Omega_{sparsity}}_{\text{sparsity regularization}},$$

Σχήμα 1: Loss Function

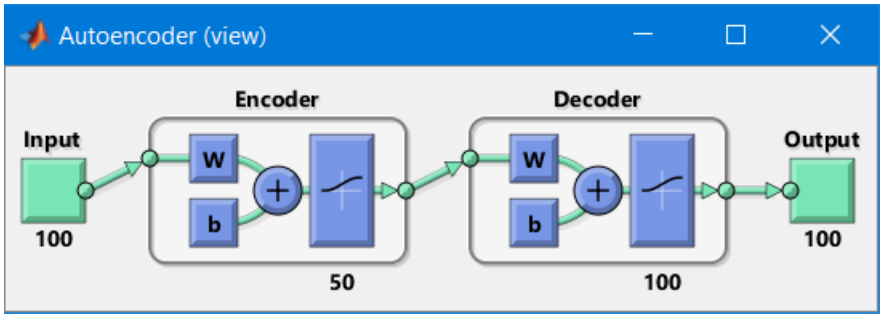
Μετά την εκπαίδευση του Autoencoder, έχουμε τη σχηματική αναπαράσταση των layer encoder και decoder καθώς και performance graphs.

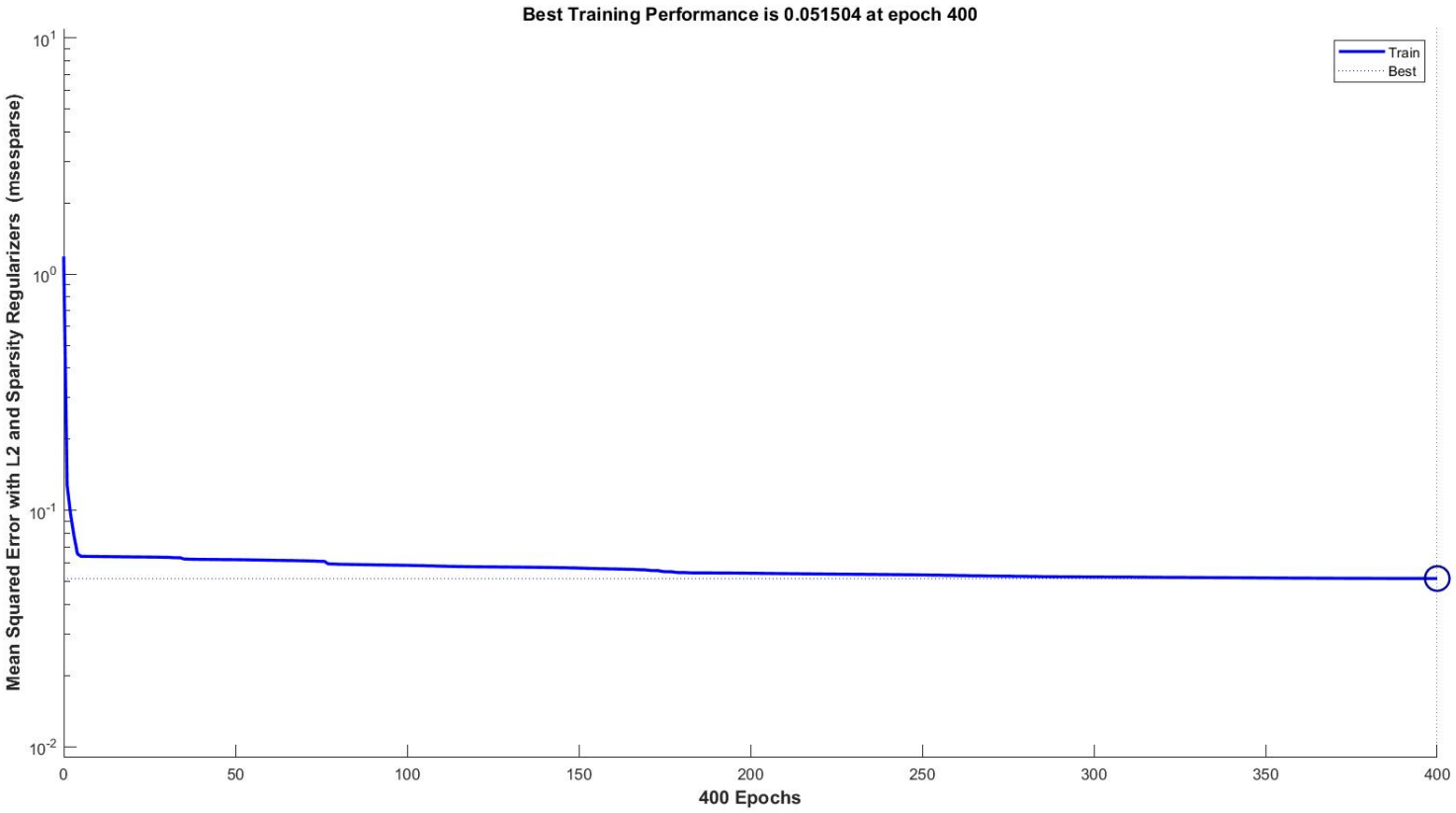
Φορτώνουμε τις test images σε ένα cell array και χρησιμοποιώντας τη συνάρτηση predict ανακατασκευάζουμε τις test images παίρνωντας ως παράμετρο τον Autoencoder. Υπολογίζουμε το μέσο τετραγωνικό σφάλμα της διαφοράς της ανακατασκευής με τα test images για να πάρουμε το ποσοστό επιτυχούς ανάκτησης.



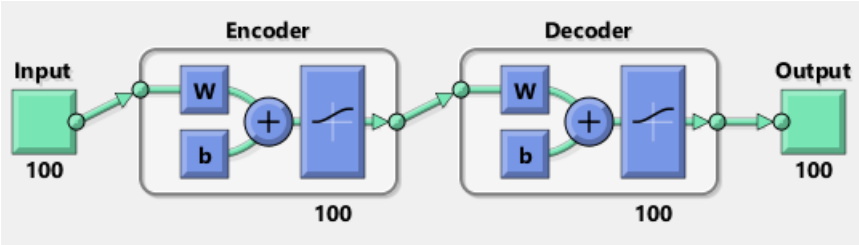


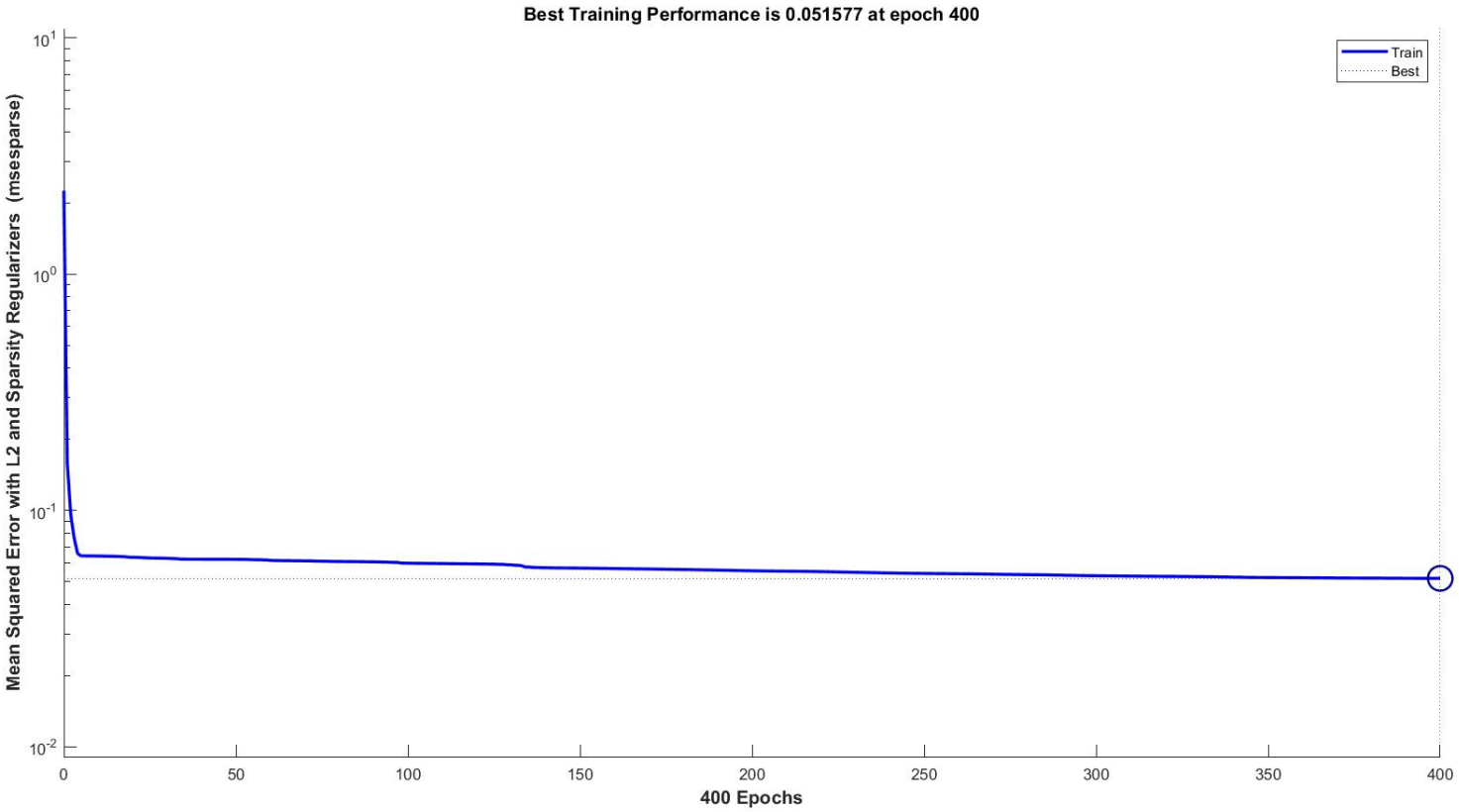
Σχήμα 2: Results for 10 eigenvectors. MSE=0.1122





Σχήμα 3: Results for 50 eigenvectors. MSE=0.1075





Σχήμα 4: Results for 100 eigenvectors. MSE=0.1089

Ας συγκρίνουμε τα αποτελέσματα ως προς:

- Accuracy: Όπως ήταν αναμενόμενο, για μικρές διαστάσεις, το PCA απέδωσε τη χαμηλότερη ακρίβεια καθώς δεν είναι σε θέση να κατανοήσει τη μη γραμμικότητα και, κατά συνέπεια, να μάθει περίπλοκους μετασχηματισμούς. Καθώς επιτρέπουμε την αύξηση της διάστασης της προβολής, η διαφορά μεταξύ κάθε μεθόδου μειώνεται. Τα PCA και τα νευρικά δίκτυα απέδωσαν συγκρίσιμη ακρίβεια καθώς έγινε πιο απλό να μάθουν μια προβολή που διατηρεί αρκετές πληροφορίες για την ταξινόμηση.
- Computation Time: Το PCA, που είναι η απλούστερη μέθοδος υπολογισμού, είναι σημαντικά ταχύτερο. Ο χρόνος εκπαίδευσης των νευρικών δικτύων εξαρτάται σε κάποιο βαθμό από τον αριθμό των epochs, ο οποίος ελέγχεται από την πρόωρη διακοπή.

Το υπολογιστικά πιο απαιτητικό μέρος του PCA είναι η ιδιογενής ανάλυση της μήτρας συνδιακύμανσης

D × D, η οποία εκτελείται χρησιμοποιώντας μια μέθοδο ισχύος στο $O(D^3)$. Η αντίστοιχη πολυπλοκότητα μνήμης του PCA είναι $O(D^2)$. Σε σύνολα δεδομένων στα οποία n<D, η υπολογιστική πολυπλοκότητα και η πολυπλοκότητα της μνήμης του PCA μπορεί να μειωθεί σε $O(n^3)$ και $O(n^2)$, αντίστοιχα. Από την άλλη, η εκπαίδευση ενός αυτόματου κωδικοποιητή χρησιμοποιώντας εκπαίδευση back propagation έχει μια υπολογιστική πολυπλοκότητα του O(inw) όπου i ο αριθμός των επαναλήψεων και w ο αριθμός των βαρών. Η εκπαίδευση των αυτόματων κωδικοποιητών μπορεί να συγκλίνει πολύ αργά, ειδικά σε περιπτώσεις όπου η διάσταση εισόδου και στόχου είναι πολύ υψηλή (καθώς αυτό αποδίδει μεγάλο αριθμό βαρών στο δίκτυο).

3 Παράρτημα

Principal Component Analysis method

```
% Script for Principal Components Analysis
% author: Evangelia Konstantopoulou
% ========================================================================

top = 10; % number of eigenvalues to consider
trainImages = 100; % number of training images
testImages = 11; % number of test images

% loading the training images to compute TrainSet which will be a 100 x
    10000 matrix
cd Database;
imFile = dir('*.jpg');
n = length(imFile);

    for j = 1:n

        filename = imFile(j).name;
        I = im2double(imread(filename));

        [X, Y, Z] = size(I);

       %turn image into grayscale if not
        if(Z == 3)
            I = rgb2gray(I);
        end

        vals = reshape(I, 1, X*Y);
        TrainSet(j, :) = vals;

    end


% subtract the mean of images in TrainSet
% calculate mean
train_mean = sum(TrainSet,1)/(trainImages);
```

```
for i = 1:trainImages
    TrainSet(i,:) = TrainSet(i,:) - train_mean;
end

% find covariance matrix
covmat = (TrainSet*TrainSet')/trainImages;
% calculate eigenvectors
[V, D] = eig(covmat);
% transEigen contains the final eigenvectors transformed into the original
   space
transEigen = TrainSet'*V;

% only keep top eigenvalues and store them in N
for i = 0:top-1
    N(:,i+1) = transEigen(:,trainImages-i);
    % normalize eigenvectors to form orthogonal space
    U = N(:,i+1);
    norm = (U'*U)^0.5;
    N(:,i+1) = N(:,i+1) / norm;
end

for i = 1:trainImages
    TrainVect(:,i) = (N)'*(TrainSet(i,:))'; % training set of reduced
       dimensional space
end

% loading the test images to compute TestSet which will be a 11 x 10000
   matrix
cd ..\test;
imFile = dir('*.jpg');
n=length(imFile);

    for j = 1:n

        filename = imFile(j).name;
        I = im2double(imread(filename));

        [X, Y, Z] = size(I);

        %turn image into grayscale if not
        if(Z == 3)
            I = rgb2gray(I);
        end

        vals = reshape(I, 1, X*Y);
        TestSet(j, :) = vals;

    end
```

```
% subtract the mean of images in TestSet
% calculate mean
test_mean = sum(TestSet,1)/testImages;

for i = 1:testImages
    TestSet(i,:) = TestSet(i,:) - test_mean;
end

for i = 1:testImages
    TestVect(:,i) = (N)'*(TestSet(i,:))'; % testing set of reduced
       dimensional space
end

for i = 1:testImages
    T1 = TestVect(:,i);

    for j = 1:trainImages
        T2 = TrainVect(:,j);
        % calculate distance between T1 and T2
        dist(j)= (T1-T2)'*(T1-T2);
    end
    % find index of sample with the shortest distance
    [~ , index] = min(dist);
    Tracker(i) = index;

end

% calculate classification accuracy
accuracy = 0;
count = 1;

    for count = 1 : 11
        if ((Tracker(count) >= count) && (Tracker(count) <= 11))
            accuracy = accuracy + 1;
        end
    end

accuracy
```

Autoencoder method

```
% Script for Autoencoder Training
% author: Evangelia Konstantopoulou
% ======================================================================

% loading the training images to compute TrainSet which will be a 100 x
    10000 matrix
```

```
cd Database;;
imFile = dir('*.jpg');
n = length(imFile);

    for j = 1:n

        filename = imFile(j).name;
        I = im2double(imread(filename));

        [X, Y, Z] = size(I);

        %turn image into grayscale if not
        if(Z == 3)
            I = rgb2gray(I);
        end

        vals = reshape(I, 1, X*Y);
        TrainSet(j, :) = vals;

    end

%train the Autoencoder
rng('default')
hiddenSize = 50;
autoenc = trainAutoencoder(TrainSet,hiddenSize, ...
    'MaxEpochs',400, ...
    'L2WeightRegularization',0.004, ...
    'SparsityRegularization',4, ...
    'SparsityProportion',0.15, ...
    'ScaleData', false);

view(autoenc)

% loading the test images to compute TestSet which will be a 11 x 10000
    matrix
cd ..\test;
f=dir('*.jpg');
files={f.name}

for k=1:numel(files)

  I=imread(files{k});
  I=im2double(I);
  TestSet{k} = rgb2gray(I);

end

TestSet=cell2mat(TestSet);
reconstructed = predict(autoenc,TestSet);
```

```
%find performance
mseError = mse(TestSet-reconstructed);
```