3η εργασία για το μάθημα
«ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ-ΒΑΘΙΑ ΜΑΘΗΣΗ»

Ονοματεπώνυμο: Χρήστος Ιωαννίδης

ΑΕΜ: 9397, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Η παρούσα εργασία αποτελεί μια αναφορά για την υλοποίηση ενός δικτύου **autoencoder** για ανακατασκευή των εικόνων χειρόγραφων ψηφίων της βάσης δεδομένων MNIST.

Το νευρωνικό δίκτυο που δημιουργήθηκε αποτελείται από δύο κομμάτια: το κομμάτι του encoder, που εκπαιδεύτηκε ώστε να μικρύνει το πλήθος της πληροφορίας της εικόνας των 28x28=784 pixels, και το κομμάτι του decoder, που παίρνοντας το μικρότερο αυτό πλήθος πληροφορίας από το τελευταίο layer του encoder, δηλαδή το λεγόμενο bottleneck, εκπαιδεύεται ώστε να κάνει την αντίστροφη διαδικασία, δηλαδή να προβλέψει και να δημιουργήσει την αρχική εικόνα.

Έτσι, ενώ σαν input έχουμε την πληροφορία των 784 pixel (αφού, φυσικά, κάνουμε reshape τον 28x28 πίνακα με την πληροφορία της κάθε εικόνας σε ένα διάνυσμα 1x784), ουσιαστικά εκπαιδεύουμε τον encoder να κρατήσει μόνο το σημαντικότερο κομμάτι της πληροφορίας (το πόσο μεγάλο η μικρό είναι αυτό εξαρτάται από το πλήθος των νευρώνων του bottleneck layer). Η πληροφορία αυτή δίνεται στον decoder και αυτός έχει σαν output αυτό που προβλέπει ως πληροφορία της εικόνας σε 784 pixel. Κάνοντας reshape, λοιπόν, αυτό το διάνυσμα, και με τη χρήση της matplotlib, μπορούμε να εμφανίσουμε την αρχική και την τελική εικόνα και να τις συγκρίνουμε.

Για το κομμάτι του κώδικα χρησιμοποιήθηκε η βιβλιοθήκη keras της tensorflow και το μοντέλου ακολούθησε δομή sequential model.

Τα στάδια φαίνονται ως εξής:

1. Κάνουμε Import τις αναγκαίες βιβλιοθήκες, φορτώνουμε τα data και κάνουμε την κανονικοποίηση για υπολογιστικούς λόγους, και το reshape όπως προαναφέρθηκε:

```
import tensorflow as tf
from tensorflow import keras
from keras.datasets import mnist
import matplotlib.pyplot as plt
#loading the mnist handwritten digits dataset, 28x28 pixel images
(x_train, y_train), (x_test, y_test)= mnist.load_data()

#we normalize the data for computational conveniece
x_train=x_train/255.0
x_test=x_test/255.0
x_train=x_train.reshape(60000,784)
x_test=x_test.reshape(10000,784)
```

2. Φτίαχνουμε το κομμάτι του encoder που φιλτράρει την πληροφορία της εικόνας:

```
autoencoder=tf.keras.models.Sequential()

#encoding phase
autoencoder.add(tf.keras.layers.Dense(128, activation="relu", input_dim=784))

#bottleneck
autoencoder.add(tf.keras.layers.Dense(32, activation="relu"))
```

3. Το κομμάτι του decoder, που στο output layer χρησιμοποιήσαμε activation function sigmoid αφού σε προηγούμενες εργασίες έδειξε ότι συμπεριφέρεται καλύτερα σε αυτό, και στη συνέχεια η δημιουργία και η εκπαίδευση του μοντέλου:

```
#decoding phase
autoencoder.add(tf.keras.layers.Dense(128, activation="relu"))
autoencoder.add(tf.keras.layers.Dense(784, activation="sigmoid"))

autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.fit(x_train, x_train, epochs=10, batch_size=256)
```

4. Ελέγχουμε το μοντέλο και κάνουμε ξανά reshape ώστε να έχουμε μορφή που θα μπορούμε να εμφανίσουμε σαν εικόνα. Τέλος, εμφανίζουμε ενδεικτικά έναν αριθμό ζευγαριών «πριν-μετά» των εικόνων:

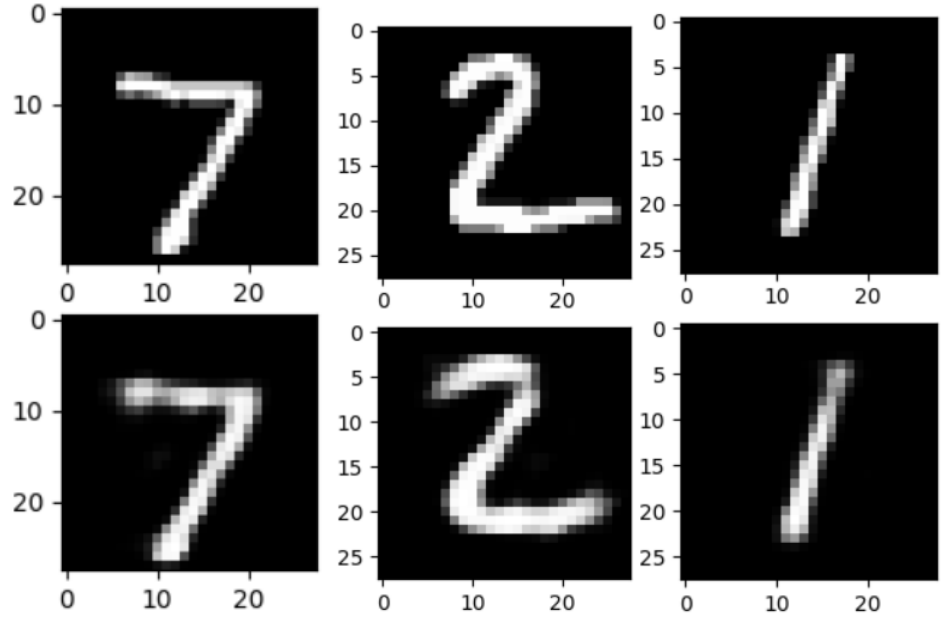
```
48  out_images=autoencoder.predict(x_test, batch_size=256)
49  out_images=out_images.reshape(10000,28,28)
50  plt.figure(figsize=(20,4))
51  x_test=x_test.reshape(10000,28,28)
52
53  ten=10
54  for i in range():
55
56   fig=plt.subplot(2,1,1)
57   plt.imshow(x_test[i], cmap="gray")
58   fig=plt.subplot(2,1,2)
59   plt.imshow(out_images[i], cmap='gray')
60   plt.show()
```

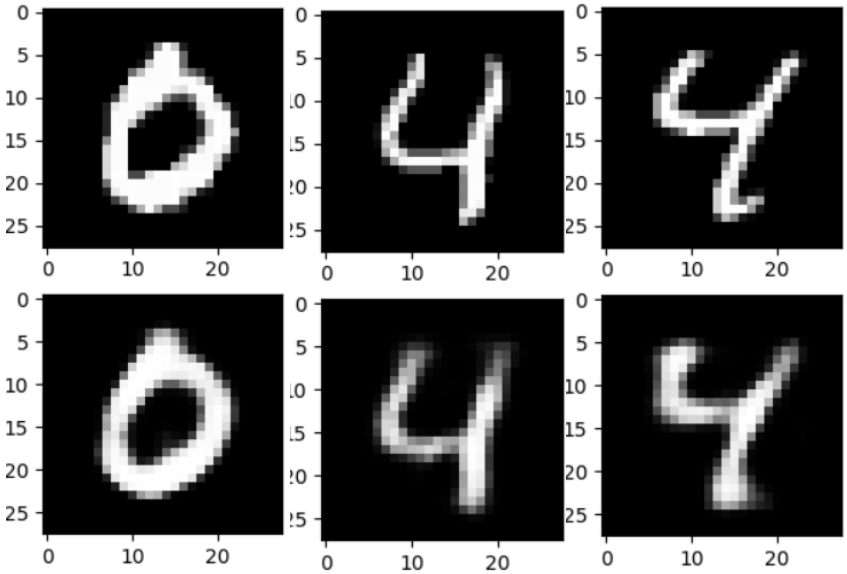
Το outcome αν τρέξουμε το πρόγραμμα δείχνει ότι έχουμε πολύ μικρά νούμερα loss και χρόνου στη διάρκεια της εκπαίδευσης:

```
Epoch 1/10
235/235 [==============================] - 2s 8ms/step - loss: 0.2286
Epoch 2/10
235/235 [==============================] - 2s 7ms/step - loss: 0.1338
Epoch 3/10
235/235 [==============================] - 2s 7ms/step - loss: 0.1167
Epoch 4/10
235/235 [==============================] - 2s 7ms/step - loss: 0.1079
Epoch 5/10
235/235 [==============================] - 2s 7ms/step - loss: 0.1034
Epoch 6/10
235/235 [==============================] - 2s 7ms/step - loss: 0.1007
Epoch 7/10
235/235 [==============================] - 2s 7ms/step - loss: 0.0988
Epoch 8/10
235/235 [==============================] - 2s 8ms/step - loss: 0.0974
Epoch 9/10
235/235 [==============================] - 2s 7ms/step - loss: 0.0963
Epoch 10/10
235/235 [==============================] - 2s 7ms/step - loss: 0.0955
```

Από τα αποτελέσματα που θα εκθέσω παρακάτω, μπορούμε να καταλάβουμε ότι παρόλο που περιορίζουμε την πληροφορία σε 32 νευρώνες, δηλαδή κρατάμε το 32/784=4.08% της πληροφορίας, αυτό είναι αρκετό για να γίνει μια αρκετά ακριβής ανακατασκευή των ψηφίων. Αυτό οφείλεται, βέβαια, εν μέρει στην ευκολία του συγκεκριμένου dataset.

Εκθέτω ενδεικτικά ορισμένα αποτελέσματα: (πάνω στήλη: ΠΡΙΝ, κάτω στήλη: ΜΕΤΑ)



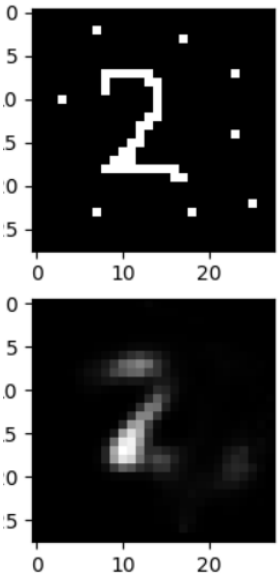


Βλέπουμε λοιπόν ότι ενώ, όπως είναι λογικό, χάνουμε σε ευκρίνεια και ορισμένες λεπτομέρειες παραλείπονται, όλοι οι αριθμοί είναι αναγνωρίσιμοι, και μπορούμε παρατηρήσουμε χαρακτηριστικά που έχουν κρατηθεί ακόμα, όπως η ουρά στο τελευταίο δείγμα με το 4, ή η γραμμούλα στην 4η φωτογραφία με το ψηφίο 0.

Έτσι, το δίκτυο μας είναι εκπαιδευμένο ώστε τώρα να μπορεί να ανταποκριθεί σε εικόνες με θόρυβο, και να κάνει μια διαδικασία “denoising”. Μπορούμε λοιπόν, τώρα, να ελέγξουμε το παραπάνω: μέσω της εφαρμογής ζωγραφικής του υπολογιστή, δημιουργούμε ένα ψηφίο σε διάσταση 28x28, και προσθέτουμε θόρυβο. Στη συνέχεια παραθέτουμε, αφού φυσικά έχουμε εκπαιδεύσει το δίκτυο μας, αυτή την εικόνα για να παρατηρήσουμε αν όντως γίνεται το denoising και ακριβής ανακατασκευή του ψηφίου μας. Ο κώδικας φαίνεται παρακάτω:

```
import cv2 as cv
import numpy as np
img=cv.imread(f'digit.png')[:,:,0]
img=np.invert(np.array([img]))
img=img/255
img=img.reshape(1, 784)
newimg=autoencoder.predict(img)
img=img.reshape(28, 28)
newimg=newimg.reshape(28, 28)
fig=plt.subplot(2,1,1)
plt.imshow(img, cmap='gray')
fig=plt.subplot(2,1,2)
plt.imshow(newimg, cmap='gray')

plt.show()
```

Και η εικόνα με το ψηφίο ΠΡΙΝ-ΜΕΤΑ φαίνεται εδώ:



Παρατηρείται ότι ενώ ο θόρυβος ήταν πυκνός σε κάποια σημεία και αυτό κατάφερε να αλλοιώσει την ποιότητα της εικόνας, να «μπερδέψει» δηλαδή τον αλγόριθμο μας, μπορούμε

και πάλι, παρόλο που έγινε απόρριψη του 96% της πληροφορίας της εικόνας στο bottleneck layer, να καταλάβουμε ότι το ψηφίο που εικονίζεται είναι το 2.

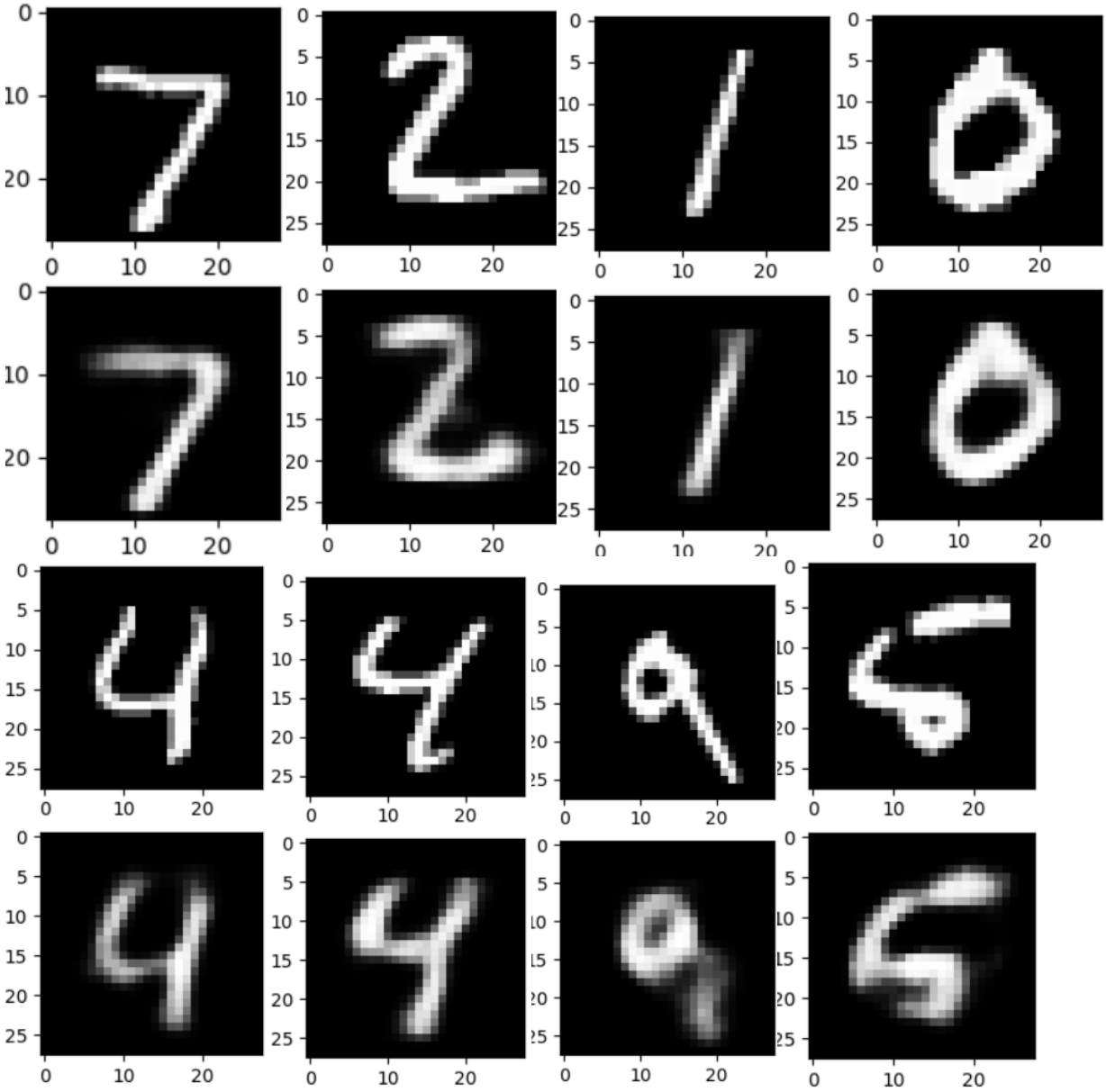
Σε αυτό το σημείο θα ήταν ενδιαφέρον να δούμε πως θα ανταποκρινόταν το δίκτυο μας αν περιορίζαμε ακόμα περισσότερο τον όγκο της πληροφορίας που μένει στο τέλος του encoder, δίνοντας στο bottleneck layer αυτή τη φορά 16 νευρώνες. Κρατάμε δηλαδή το 2% της πληροφορίας της κάθε εικόνας:

```
27    #bottleneck
28    autoencoder.add(tf.keras.layers.Dense(16, activation="relu"))
```

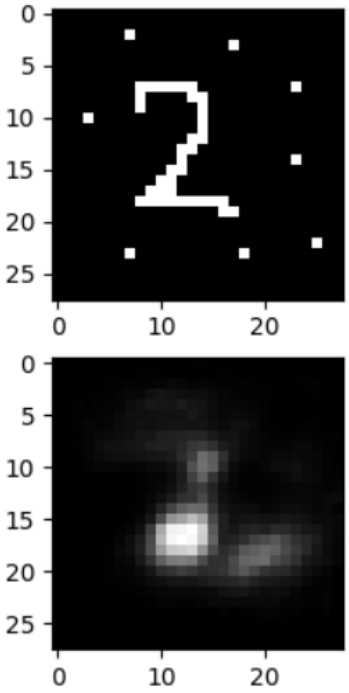
Από το outcome παρατηρούμε ότι, όπως ήταν αναμενόμενο, στην εκπαίδευση έχουμε μεγαλύτερο loss.

```
Epoch 1/10
235/235 [==============================] - 2s 7ms/step - loss: 0.2448
Epoch 2/10
235/235 [==============================] - 2s 8ms/step - loss: 0.1470
Epoch 3/10
235/235 [==============================] - 2s 8ms/step - loss: 0.1316
Epoch 4/10
235/235 [==============================] - 2s 8ms/step - loss: 0.1264
Epoch 5/10
235/235 [==============================] - 2s 8ms/step - loss: 0.1235
Epoch 6/10
235/235 [==============================] - 2s 7ms/step - loss: 0.1215
Epoch 7/10
235/235 [==============================] - 2s 8ms/step - loss: 0.1199
Epoch 8/10
235/235 [==============================] - 2s 7ms/step - loss: 0.1185
Epoch 9/10
235/235 [==============================] - 2s 7ms/step - loss: 0.1173
Epoch 10/10
235/235 [==============================] - 2s 7ms/step - loss: 0.1162
```

Όσον αφορά τις εικόνες που προκύπτουν, αυτές φαίνονται παρακάτω, και είναι ακόμα και τώρα δυνατό να διακριθεί το ψηφίο, αν και φυσικά χάνουμε αρκετά περισσότερες λεπτομέρειες, όπως φαίνεται στις ανακατασκευές δειγμάτων που είχαμε και πριν:



Όσον αφορά την περίπτωση του ψηφίου 2 με θόρυβο, ο αλγόριθμος μας το ανακατασκεύασε με αρκετά μικρότερη επιτυχία, στο σημείο που είναι δύσκολο να διακρίνεις. Με άλλα λόγια, με τον περιορισμό της πληροφορίας σε τόσο μεγάλο ποσοστό, είναι δύσκολο να γίνει αποτελεσματικά το denoising.



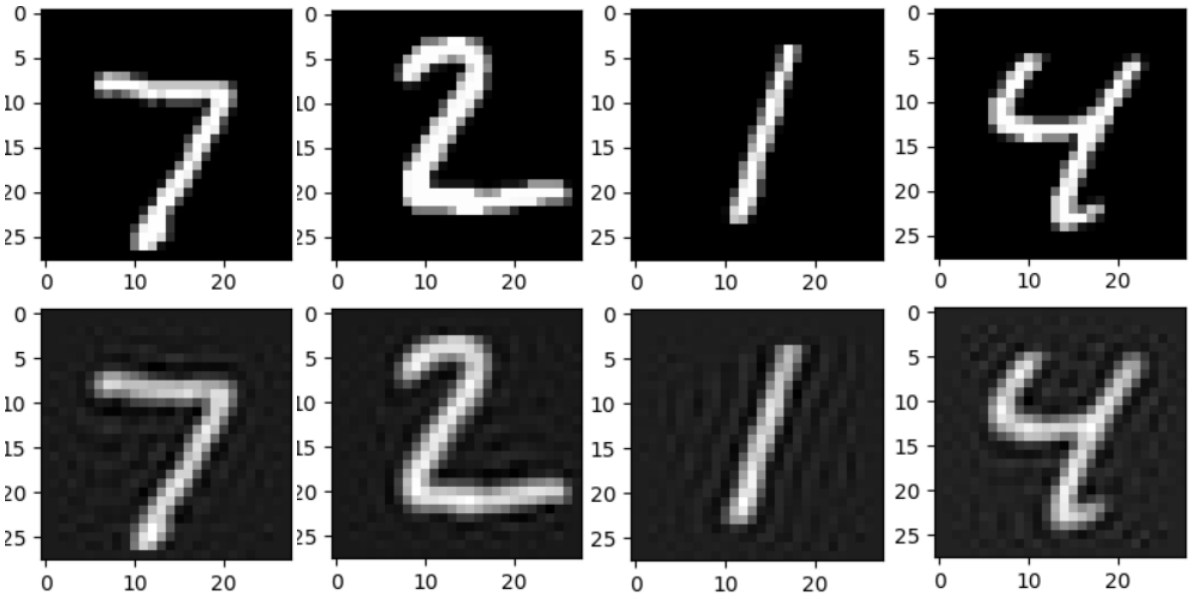
Τέλος, την ανακατασκευή μέσω PCA, που κάνει μια παρόμοια διαδικασία: μειώνει τις διαστάσεις, κάνοντας όμως maximize το variance στα δεδομένα. Τρέχουμε τον κώδικα για ποσοστό 95% της πληροφορίας:

```
import tensorflow as tf
from tensorflow import keras
from keras.datasets import mnist
import matplotlib.pyplot as plt

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler



#loading the mnist handwritten digits dataset, 28x28 pixel images
(x_train, y_train), (x_test, y_test)= mnist.load_data()

#we normalize the data for computational conveniece
x_train=x_train/255.0
x_test=x_test/255.0
x_train=x_train.reshape(60000,784)
x_test=x_test.reshape(10000,784)

#PCA
scaler = StandardScaler()
scaler.fit(x_train)
x_train=scaler.transform(x_train)
x_test1=scaler.transform(x_test)
pca = PCA(.95)
pca.fit(x_train)
x_train = pca.transform(x_train)
x_test1 = pca.transform(x_test)
```

Και κάνουμε plot τις ανακατασκευασμένες εικόνες μαζί με τις καινούριες για να συγκρίνουμε το πριν-μετά:

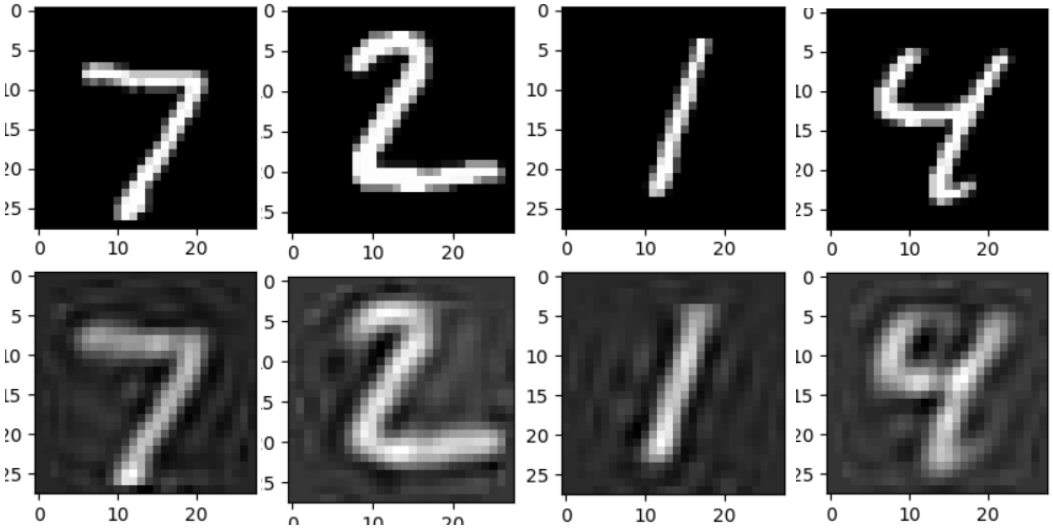
```
ten=10
for i in range(ten):

 fig=plt.subplot(2,1,1)
 plt.imshow(x_test[i].reshape(28,28), cmap="gray")
 fig=plt.subplot(2,1,2)
 plt.imshow(newx_test[i].reshape(28,28), cmap='gray')
 plt.show()
```

Τα αποτελέσματα:



Έχουμε πολύ καλή αναπαράσταση της λεπτομέρειας, αν και πάλι χάνουμε λίγο σε ευκρίνεια. Είναι λογικό να έχουμε τόσο ακριβή ανακατασκευή αφού κρατήσαμε το 95% της πληροφορίας.

Δοκιμάζουμε άλλη μια φορά, για 75% της πληροφορίας:



Οπού έχουμε αποτελέσματα, τέτοια ώστε λεπτομέρειες να φαίνονται καλύτερα στην μέθοδο του autoencoder. Τέλος θα δοκιμάσουμε να τρέξουμε το PCA στην εικόνα που δημιούργησα με το θόρυβο, συμπληρώνοντας τον κώδικα με το 75% με αυτό:

```
import cv2 as cv
import numpy as np
img=cv.imread(f'digit.png')[:,:,0]
img=np.invert(np.array([img]))
img=img/255
img=img.reshape(1, 784)
img1=pca.transform(img)
pcaim=pca.inverse_transform(img1)
plt.imshow(pcaim.reshape(28,28), cmap="gray")
plt.show()
```

Και προκύπτει ένα αποτέλεσμα που φαίνεται ότι δεν κάνει τόσο καλό denoising όσο ο autoencoder μας: