Νευρωνικα Δικτυα - Εργασια 3

Ρούσος Σταμάτης

22 Ιανουαρίου 2026

Εισαγωγή

Σε αυτη την εργασια υλοποιησα ενα RBF neural network με σκοπο την κατηγοριοποιηση των εικονων στο CIFAR-10 Dataset, και Autoencoder με σκοπο την ανακατασκευη εικονας. Συγκεκριμενα υλοποιησα:

- RBF:
  1. RBF με ετοιμη υλοποιηση kmeans της sklearn για τα κεντρα οπου έκανα grid search για το σ
  2. RBF με δικη μου υλοποιηση kmeans
  3. RBF με κεντρα τυχαια training data
  4. CNN ως feature extractor χρησιμοποιωντας προεκπαιδευμενο CNN απο την πρωτη εργασία + RBF για classification
- Autoencoder:
  1. Διαφορες αρχιτεκτονικες μοντελων Autoencoder με σκοπο την ευρεση αυτου που κανει την καλυτερη ανακατασκευη εικονας
  2. PCA για ανακατασκευη εικονας και συγκριση
  3. Encoder + CNN για classification

  (α□) Με pre-trained παγωμενο encoder

  (β□) Με pre-trained learnable encoder

4. Encoder + SVM για classification
5. Denoising Autoencoder στο STL-10

RBF

Τα RBF νευρωνικα δικτυα αποτελουνται συνηθως απο ενα κρυφο επιπεδο. Ειναι δικτυα που μετρουν ομοιοτητες μεταξυ πραγματων. Συγκεκριμενα:

- Χρησιμοποιουν κεντρα:
  Καθε νευρωνας στο κρυφο τους επιπεδο αντιστοιχει σε ενα κεντρο C_i
- Οι συναρτησεις ενεργοποιησης τους ειναι:

$$\phi_i(x) = e^{-\frac{||x-C_i||^2}{2\sigma^2}}$$

Οσο πιο "κοντα" ειναι το δειγμα εισοδου στο κεντρο C_i του νευρωνα i, δηλαδη οσο περισσοτερο μοιαζει με αυτο, τοσο περισσοτερο ενεργοποιειται η ϕ_i (παιρνει μεγαλυτερες τιμες).

Για παραδειγμα:

$$\text{Αν } x = C_i \Rightarrow \phi_i(x) = e^0 = 1$$

$$\text{Αν } ||x - C_i|| \to \infty \Rightarrow \phi_i(x) \to 0$$

Η εξοδος του RBF ειναι οι ενεργοποιησεις:

$$\phi(x) = [\phi_1(x), \phi_2(x), \ldots, \phi_M(x)]$$

που μας δειχνουν ποσο κοντα ειναι το x σε καθε κεντρο.

Στη δικη μου εργασια, ηθελα να χρησιμοποιησω RBF για να κανω κατηγοριοποιηση.

Επομενως, στην εξοδο του RBF layer προσθεσα εναν κατηγοριοποιητη, ο οποιος ειναι στην ουσια σαν τη softmax. Ο κατηγοριοποιητης αυτος ειναι ο logistic regression και στην ουσια ειναι

ενα Dense 10 layer + softmax layer. Μαθηματικα:

$$z = W\phi(x) + b$$

$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \quad i = 1, \ldots, K$$

Η αποφαση:

$$\hat{c} = \arg\max_i \hat{y}_i$$

Πορεια υλοποιησης RBF

- Data preprocessing
  - Normalize data (pixel values / 255)
  - Reshape data
  - Create validation set
  - Scaling ($z = \frac{x-\mu}{\sigma}$)
  - PCA
  - Subset λογω RAM
- KMeans (sklearn)
  2000 κεντρα
- RBF + Logistic Regression classifier
  Πρωτα Grid search για 5 διαφορετικα σ
  Μετα στενοτερο grid search για 5 διαφορετικα σ
  Τελικο RBF με το σ με το μεγαλυτερο validation accuracy
- Υλοποιηση δικου μου kmeans

Από θεωρια γνωριζουμε οτι ο kmeans κανει βελτιστοποιηση σε 2 βηματα:

1. Αρχικα αρχικοποιει τα κεντρα του ως καποια τυχαια training samples

2. Κατοπιν κανει clustering:

   - Για καθε δειγμα εκπαιδευσης υπολογιζει την αποσταση απο καθε κεντρο
   - Για το δειγμα x_n, ψαχνει το min $||x_n - C_k||^2$
   - Το δειγμα ανηκει στο cluster k του πιο κοντινου κεντρου C_k

3. Τελος κανει ενημερωση των κεντρων:

   - Βρισκει νεα κεντρα, παιρνοντας τα δειγματα που ανηκουν σε καθε cluster, και για καθε cluster ψαχνει το νεο κεντρο του, ως αυτο που ειναι πιο κοντα σε ολα του τα δειγματα. Δηλαδη, για τα x_n που ανηκουν στο cluster k, ψαχνει το $c_{k,new}$ που ελαχιστοποιει:

$$\min_{c_k} \sum_{x_n \in clusterk} ||x_n - c_k||^2$$

     Προκυπτει, οτι λυση αυτου του προβληματος ειναι ο μεσος ορος των δειγματων του cluster k.

$$C_k^{new} = \frac{1}{N_k} \sum_{x_n \in cluster_k} x_n$$

Η διαδικασια επαναλαμβανεται μεχρι τα κεντρα να μην αλλαζουν.

Το συνολικο προβλημα γραφεται:

$$\min_{Z_{nk}, C_k} \sum_{n=1}^{N} \sum_{k=1}^{K} Z_{nk} ||x_n - C_k||^2$$

με περιορισμους:

$$Z_{nk} \in \{0, 1\}, \quad \sum_{k=1}^{K} Z_{nk} = 1$$

Το προβλημα δεν ειναι κυρτο, οποτε σπαει σε αυτα τα 2 επιμερους υποπροβληματα που ειναι κυρτα και τα λυνει εναλλασσομενα, προκειμενου να επιλυθει, .

Ετσι λοιπον εφτιαξα 4 συναρτησεις:

- Η 1η αρχικοποιει τυχαια τα κεντρα στο πρωτο βημα, Θετοντας τα ισα με καποια τυχαια training samples. Επιστρεφει τα κεντρα σε ενα πινακα $(, F)$, οπου K το πληθος των κεντρων και F τα features των training data.
- Η 2η κανει το clustering. Υπολογιζει τις αποστασεις καθε δειγματος απο καθε κεντρο, και τοποθετει το δειγμα στο cluster του κοντινοτερου κεντρου. Ο πινακας αποστασεων ειναι διαστασεων (N, K). Επιστρεφει το διανυσμα labels $(N,)$
- Η 3η ενημερωνει τα κεντρα. Παιρνει τα δειγματα που ανηκουν στο cluster j και βρισκει το νεο κεντρο τους ως το μεσο ορο τους. Επιστρεφει τα νεα κεντρα ως πινακα (K,F)
- Η 4η συναρτηση, αφου καλεσει την 1η για αρχικοποιηση, επαναλαμβάνει την 2η και 3η, και σταματαει οταν τα κεντρα μετακινουνται ελαχιστα.

Στη συνεχεια υλοποιησα RBF χωρις kmeans χρησιμοποιωντας ως κεντρα τυχαια training samples. Αυτη η μεθοδος ηταν πολυ πιο γρηγορη και μαλιστα σε ακριβεια ηταν οριακα καλυτερη απο RBF με kmeans.

Τελος χρησιμοποιησα το εκπαιδευμενο CNN της πρωτης μου εργασιας ως feature extractor, βγαζοντας το classification (Dense 10 + softmax), και εδωσα την εξοδο του στο RBF για classification. Τα αποτελεσματα ηταν πολυ καλυτερα.

Αποτελέσματα

Για τα πρωτα 9 μοντελα χρησιμοποιηθηκε kmeans της sklearn με 2000 κεντρα τα οποια χρειαστηκαν 67.22 sec για να υπολογιστουν.

Μοντέλο	σ	val-acc	total time
1	20	0.4132	24.54
2	30	0.4401	64.34
3	35	0.4410	84.85
4	40	0.4379	98.64
5	45	0.4351	129.90
6	50	0.4311	152.32
7	60	0.4203	178.38
8	80	0.4038	178.63
9	200	0.3883	274.98

Το καλυτερο μοντελο που κρατησα ηταν:

Μοντέλο	σ	val-acc	test-acc	total time
4	40	0.4379	0.4407	126.07

Το επομενο μοντελο χρησιμοποιει τη δικη μου υλοποιηση kmeans με 200 κεντρα (με παραπανω επαιρνε παρα πολλη ωρα). Τα κεντρα χρειαστηκαν 280.7 sec (4.7 λεπτα) για να υπολογιστουν.

Μοντέλο	σ	val-acc	test-acc	total time
10	40	0.3833	0.3839	10.34

Το επομενο μοντελο χρησιμοποιει ως κεντρα 2000 τυχαια training samples. Ο χρονος ευρεσης των κεντρων ηταν, προφανως, ακαριαιος:

Μοντέλο	σ	val-acc	test-acc	total time
11	40	0.448	0.4458	78.98

Παραδοξως εδωσε τα καλυτερα αποτελεσματα μεχρι τωρα.

Το τελευταιο μοντελο χρησιμοποιει το εκπαιδευμενο CNN ως feature extractor. Τα κεντρα ηταν 300 και υπολογιστηκαν με kmeans της sklearn σε 28.90 sec.

Μοντέλο	σ	val-acc	test-acc	total time
12	30	0.831	0.8032	17.4

Παρατηρήσεις

- Η εργασια εγινε εξ ολοκληρου στο Colab
- Σε ολα τα μοντελα χρησιμοποιηθηκε training subset των 10.000 εικονων για τα πρωτα 11, και των 12.000 για το τελευταιο, διοτι εσκαγε η RAM.
- Χρησιμοποιησα PCA για τη συμπιεση των διαστασεων.
- Η αποδοση του RBF στην κατηγοριοποιηση μου φανηκε αρκετα χαμηλη (max 44%) και δεν ειμαι σιγουρος γιατι. Προφανως, οσο περισσοτερα κεντρα βαλω (και training samples) τοσο καλυτερα αποτελεσματα θα εχω (μεχρι ενα σημειο).
- Δεν καταλαβαινω πως το RBF με τυχαια κεντρα εδωσε οριακα καλυτερα αποτελεσματα απο RBF με kmeans με το ιδιο πληθος κεντρων. Υποτιθεται πως ο kmeans βρισκει "καλυτερα" κεντρα.
- Στην αρχη του notebook ειχα ξεχασει να κανω scaling και επαιρνα πολυ χειροτερα αποτελεσματα.
- Για να τρεξει το CNN στο τελος, παραθετω το αρχειο με τα εκπαιδευμενα βαρη "final model6.h5"το οποιο εγω ανεβαζα χειροκινητα στο Colab.

Autoencoder

Οι Autoencoders αποτελουνται απο 3 κομματια, τον encoder, τον decoder και το Bottleneck ή αλλιως Latent Space. Ενα μεγαλο κομματι εφαρμογης τους ειναι η ανακατασκευη εικονων. Στην παρουσα εργασια, χρησιμοποιηθηκαν για αυτον ακριβως το λογο. Η πορεια που ακολουθησα ειναι η εξης:

- Αρχικα εψαξα την καλυτερη αρχιτεκτονικη Autoencoder που μου δινει το μικροτερο loss στην ανακατασκευη εικονων. Δοκιμασα διαφορετικους learning rates, διαφορετικα losses, μειωση Height και Width και channels εικονας μεχρι το Latent Space, μειωση Height/Width αλλα αυξηση channels, Convolutional Autoencoder, Dense Autoencoder, διαφορετικα Layers στον Decoder οπως Conv2D + Upsampling / Conv2DTranspose με strides για αυξηση διαστασεων κλπ. Παρατηρησα πως οσο περισσοτερο ριχνω τις διαστασεις στο Latent Space, τοσο πιο κακη η ανακατασκευη. Επομενως, κρατησα ψηλα τις διαστασεις στο Latent Space (2048 απο 3072).

- Εφτιαξα Learning Curves για το καλυτερο μοντελο και το αξιολογησα στο test set.

- Χρησιμοποιησα PCA (το οποιο στην ουσια ειναι ενας Autoencoder με γραμμικες ενεργοποιησεις) για ανακατασκευη validation data, κανοντας fit στα training data, transform και μετα inverse transform στα validation data και μετρησα το loss. Παραδοξως, οι εικονες εμφανιστηκαν πολυ καλυτερες απο ολους τους Autoencoders το οποιο φαινεται και απο το loss, κατι το οποιο δεν το περιμενα. Σε ολους τους Autoencoders, το Output layer χρησιμοποιει sigmoid activation ωστε να δινει εξοδο στο διαστημα [0,1] καθως στην ανακατασκευη εικονας η εξοδος ειναι pixel values, οποτε με αυτο τον τροπο ειναι κανονικοποιημενες. Στο PCA δεν γινοταν αυτο, οποτε επρεπε να προσεξω να κανω clip την εξοδο στο [0,1] (δηλαδη οποια τιμη ειναι κατω απο το 0 να γινεται 0 και οποια πανω απο 1 να γινεται 1). Ωστοσο, επειδη δοκιμασα και χωρις clip, η διαφορα ηταν ελαχιστη που σημαινει οτι ειτε ελαχιστες τιμες πεφτουν εκτος διαστηματος στην ανακατασκευη του PCA, ειτε γινεται clipping εσωτερικα.

- Στη συνεχεια, χρησιμοποιησα μονο τον encoder του καλυτε-

ρου Autoencoder για μειωση διαστασεων στο Latent Space, και εδωσα τα latent space data στο CNN της 1ης εργασιας μου για classification. Στην 1η περιπτωση παγωσα τα βαρη του encoder ωστε να μην εκπαιδευονται καθολου κατα την εκπαιδευση του συνολικου μοντελου (encoder-cnn), ενω στην 2η περιπτωση αφησα τα βαρη του encoder να μπορουν να εκπαιδευτουν μαζι με το συνολικο μοντελο. Τα αποτελεσματα ηταν αρκετα καλυτερα στην 2η περιπτωση, ομως με σημαντικο overfitting. Σχεδιασα Learning Curves και confusion Matrix για το συνολικο μοντελο.

- Κατοπιν, εφαρμοσα την ιδια ακριβως τεχνικη, απλως αντι για CNN εβαλα ενα γραμμικο SVM για classification. Δουλεψα σε training subset των 9000 εικονων για το SVM γιατι αλλιως δεν εβγαζε αποτελεσμα (το ειχα αφησει 1 ωρα να εκπαιδευεται και τελικα εσκασε).
- Τελος, εφτιαξα εναν Denoising Autoencoder. Δηλαδη, προσθεσα θορυβο στις εικονες, και σκοπος του Autoencoder ειναι να τις ανακατασκευασει καθαρες. Εδω χρησιμοποιησα το STL-10 Dataset που εχει εικονες υψηλοτερης ευκρινειας, ωστε να βλεπουμε καθαρα το θορυβο και τη βελτιωση της εικονας (στο cifar 10 ειναι ηδη πολυ θολες).

Τα Losses που χρησιμοποιησα ειναι MSE (Mean - Square - Error) και Binary crossentropy.

MSE:

$$\mathcal{L}_{\text{MSE}}(x, \hat{x}) = \frac{1}{N} \sum_{i=1}^{N} (x_i - \hat{x}_i)^2$$

Binary Crossentropy (Μονο σε 1 μοντελο):

$$\mathcal{L}_{\text{BCE}}(x, \hat{x}) = -\frac{1}{N} \sum_{i=1}^{N} [x_i \log(\hat{x}_i) + (1 - x_i) \log(1 - \hat{x}_i)]$$

Το μοντελο προσπαθει να ελαχιστοποιησει το Loss, δηλαδη οπως βλεπουμε απο το $L_{MSE}(x, \hat{x})$, ιδανικα:

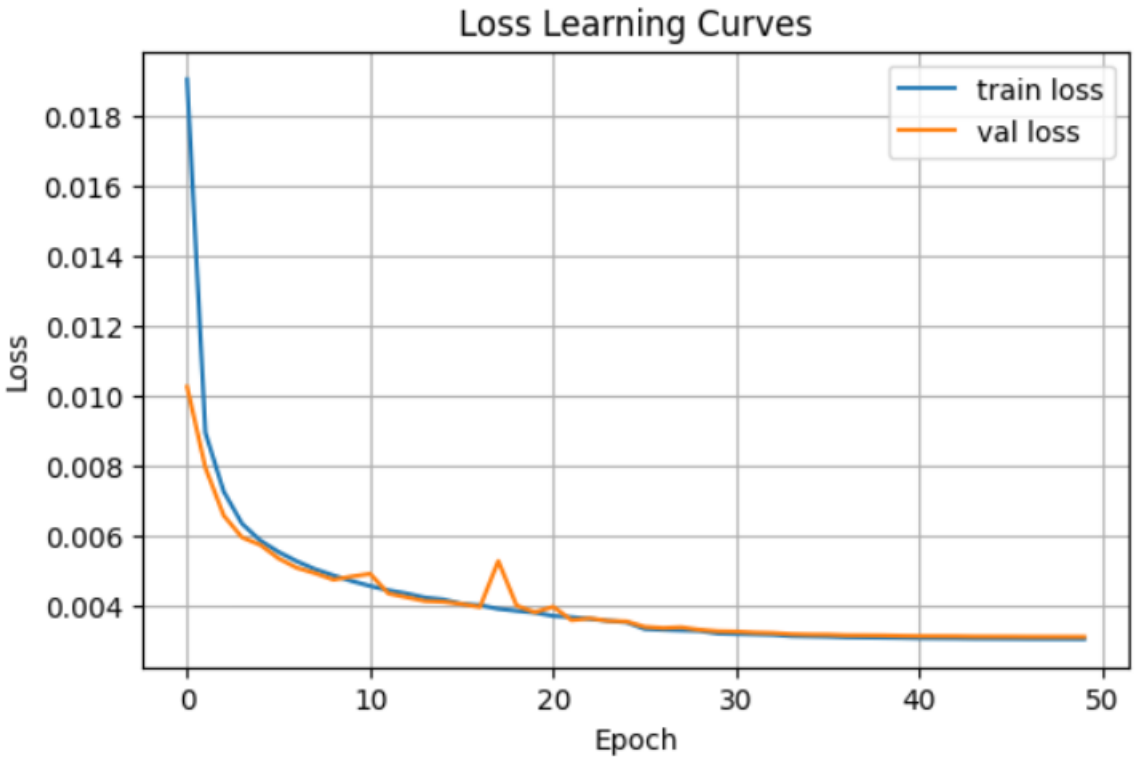
$$\hat{x} \approx x \iff L_{MSE}(x, \hat{x}) \approx 0$$

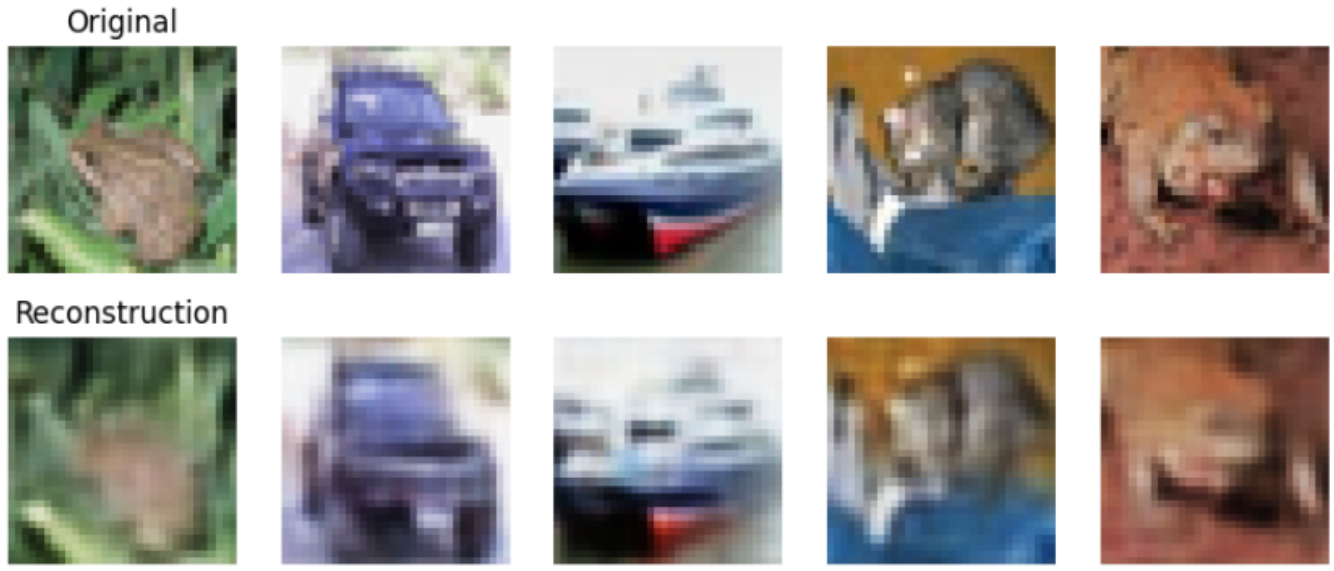
Αποτελέσματα

Μοντέλο	Latent Dim	loss	lr	val-loss	total time
1	128	MSE	0.0001	0.0111	-
2	2048	MSE	0.0001	0.0045	336.02
3	2048	MSE	0.0005	0.0038	349.01
4	2048	MSE	0.001	0.0031	253.04
5	2048	Binary Cross	0.001	0.5539	255.0
Dense	512	MSE	0.0001	0.0044	88.96

Ως καλυτερο μοντελο κρατησα το μοντελο 4. Αποτελεσματα:

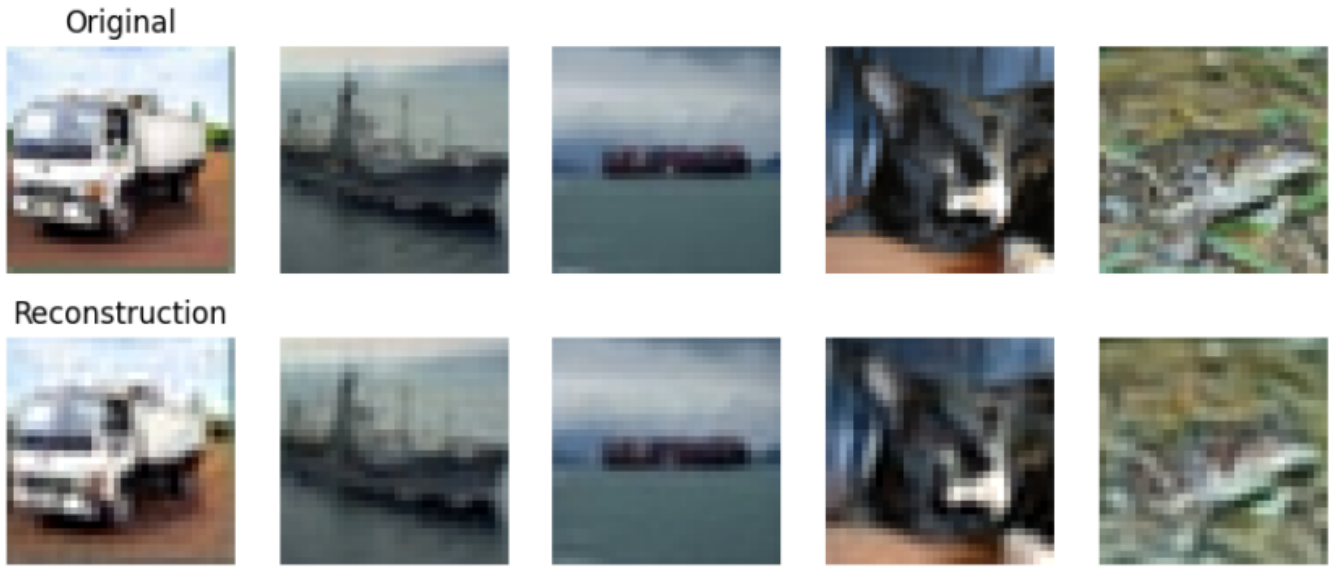
Μοντέλο	Latent Dim	loss	lr	val-loss	test-loss
4	2048	MSE	0.001	0.0031	0.0031





7

Το επομενο μοντελο ειναι το PCA. Το εξτρα βημα που εκανα εδω απο τη χρηση του PCA που καναμε παντα, ειναι οτι μετα την μειωση της διαστασης με την pca fit transform, χρησιμοποιησα την pca inverse transform για την ανακατασκευη της εικονας.

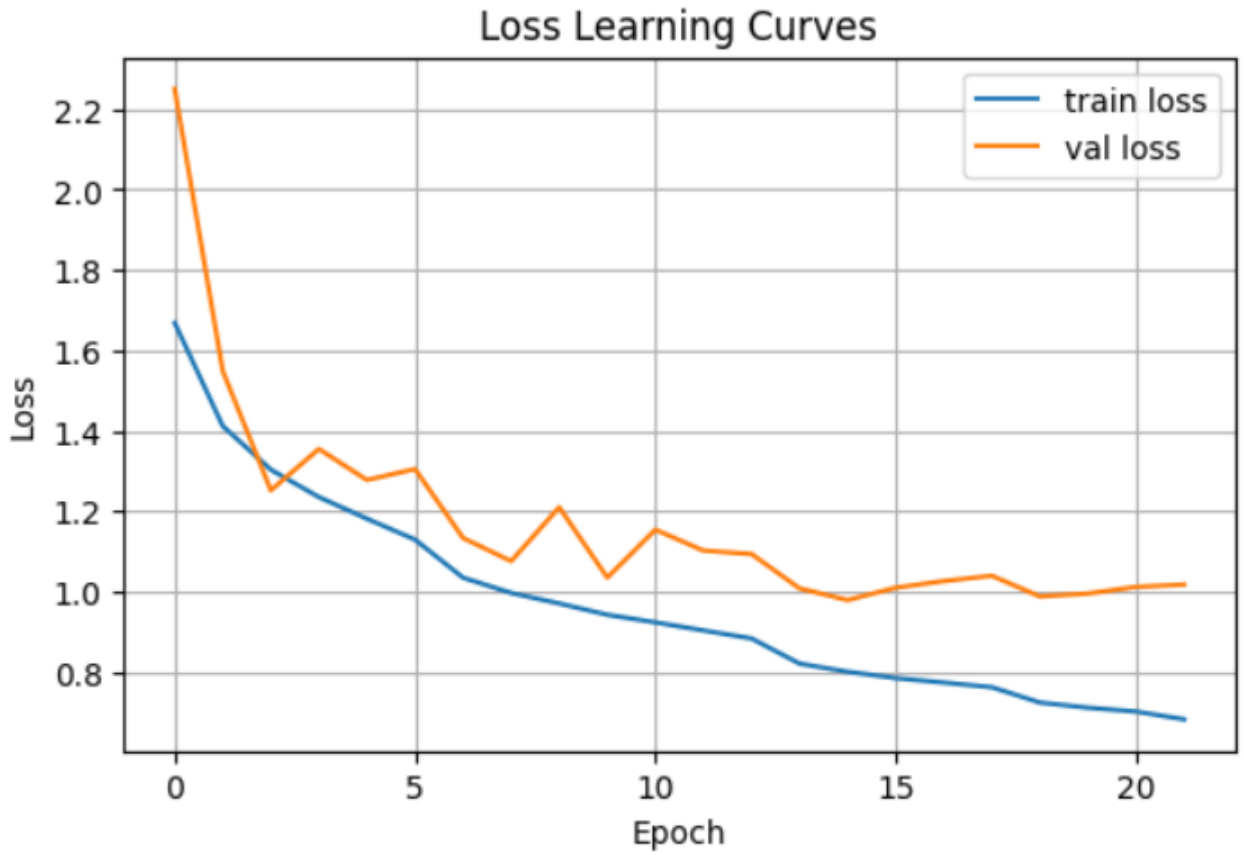


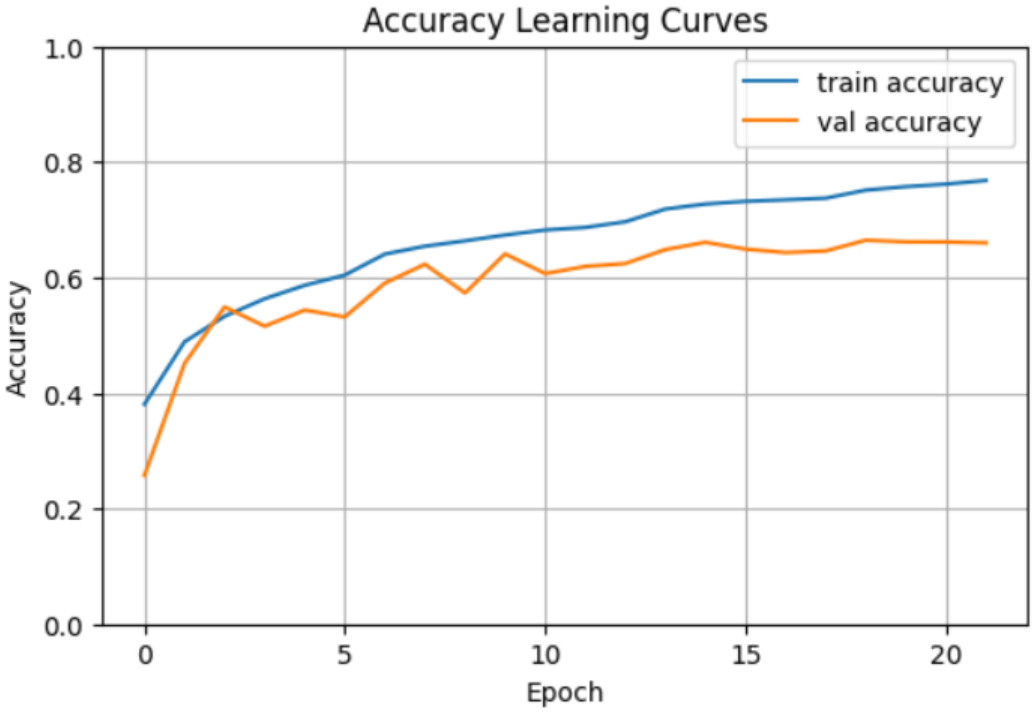
8

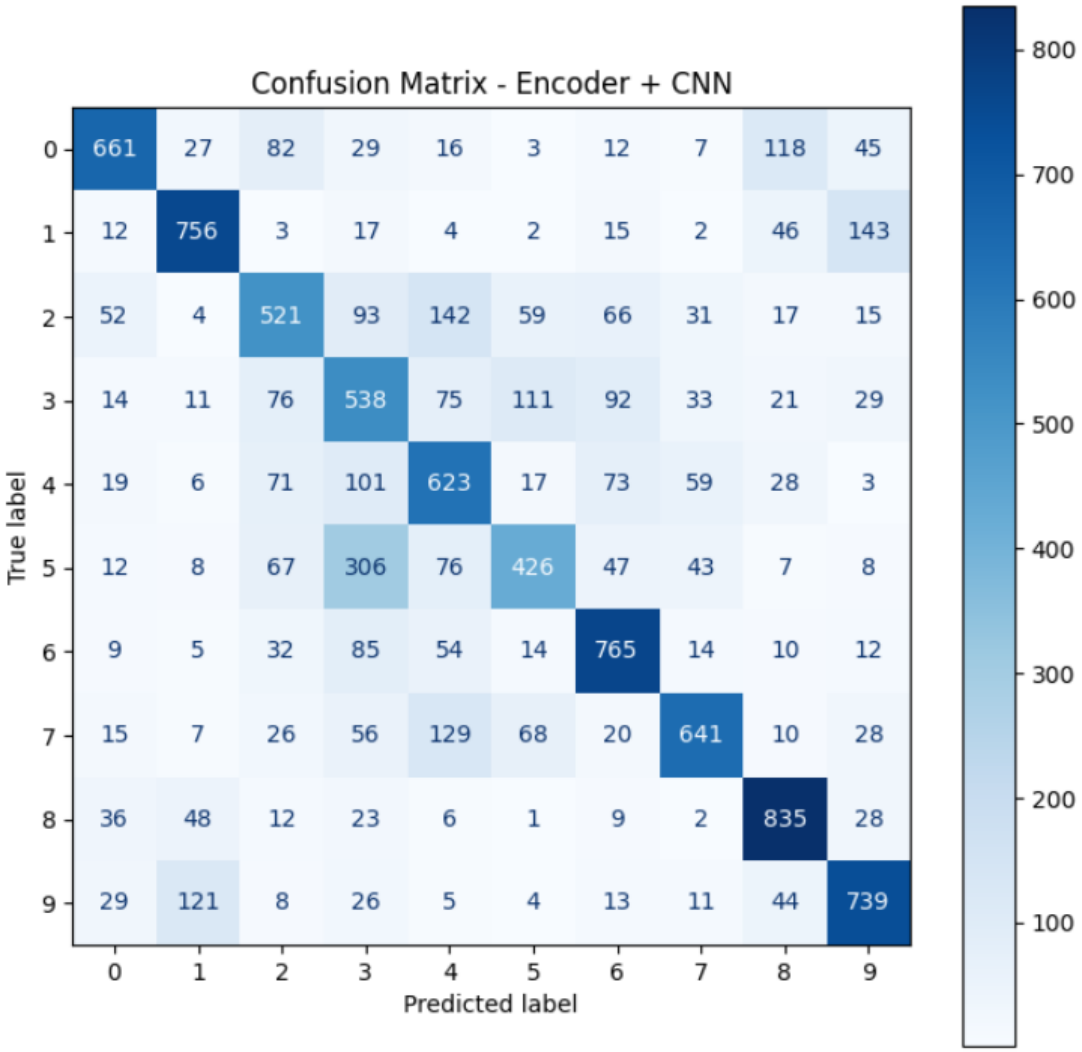
Τα επομενα μοντελα ειναι Encoder + CNN για classification. Εφτιαξα καινουργιο CNN, καθως το CNN που ειχα στην 1η εργασια δεχεται σαν εισοδο (32,32,3) εικονες, ενω ο encoder βγαζει στην εξοδο του latent space (4,4,128) εικονες. Ωστοσο η αρχιτεκτονικη ηταν ολοιδια. Οι παρακατω μετρικες αφορουν την κατηγοριοποιηση.

8a

Μοντέλο	Encoder	val-acc	test-acc
Encoder + CNN	Non-Trainable	0.661	0.65

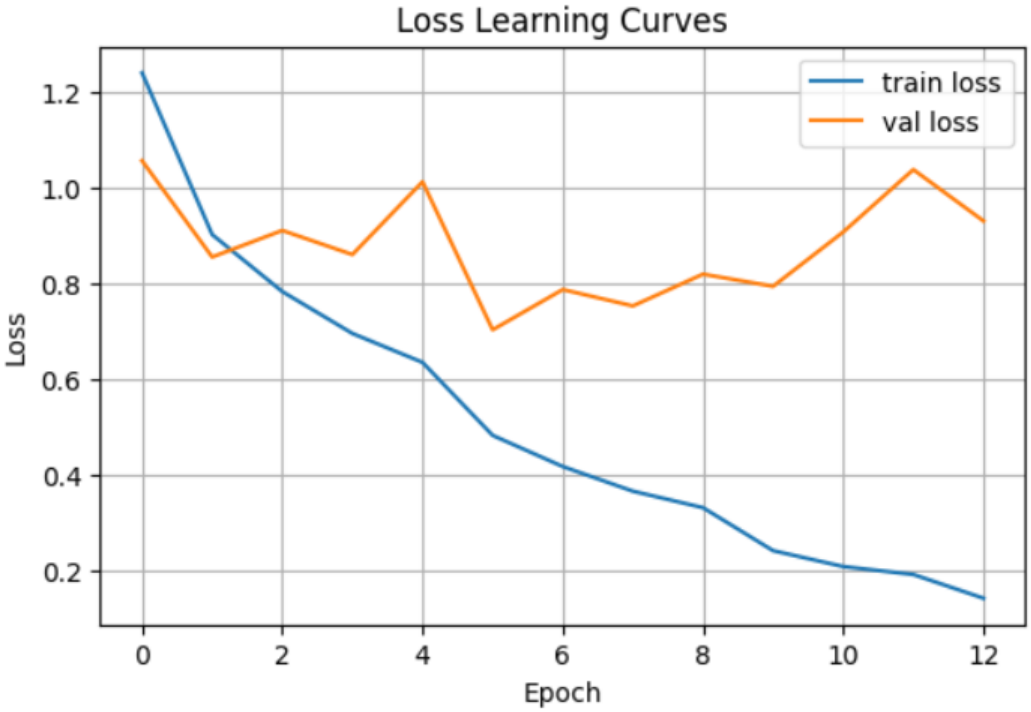


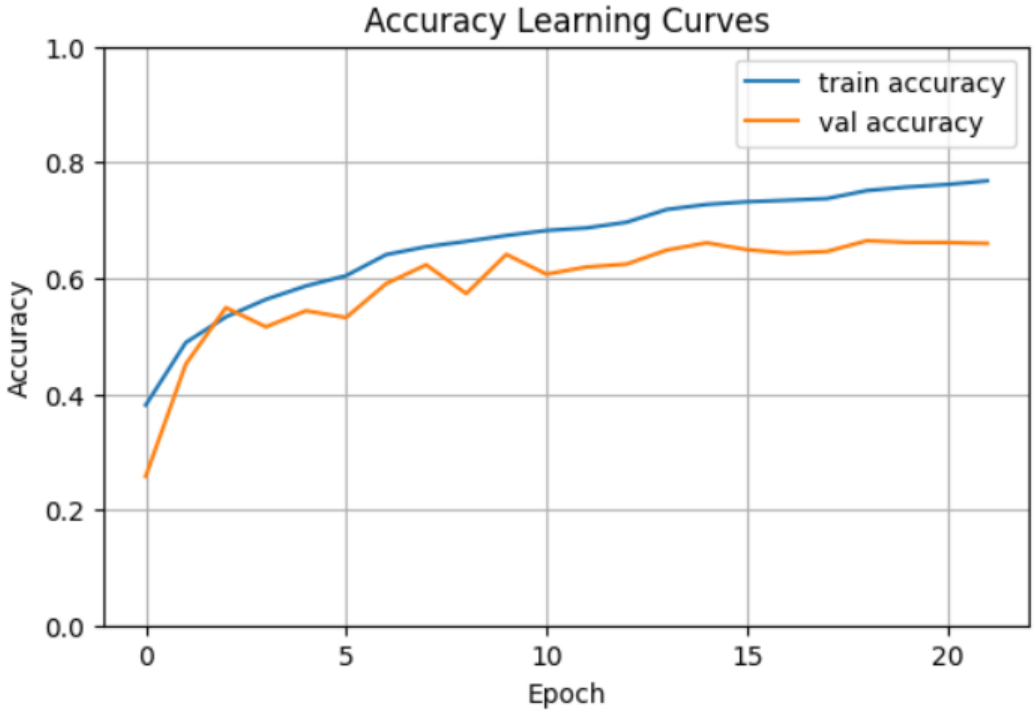

Accuracy Learning Curves
train accuracy
val accuracy
Accuracy
Epoch
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20

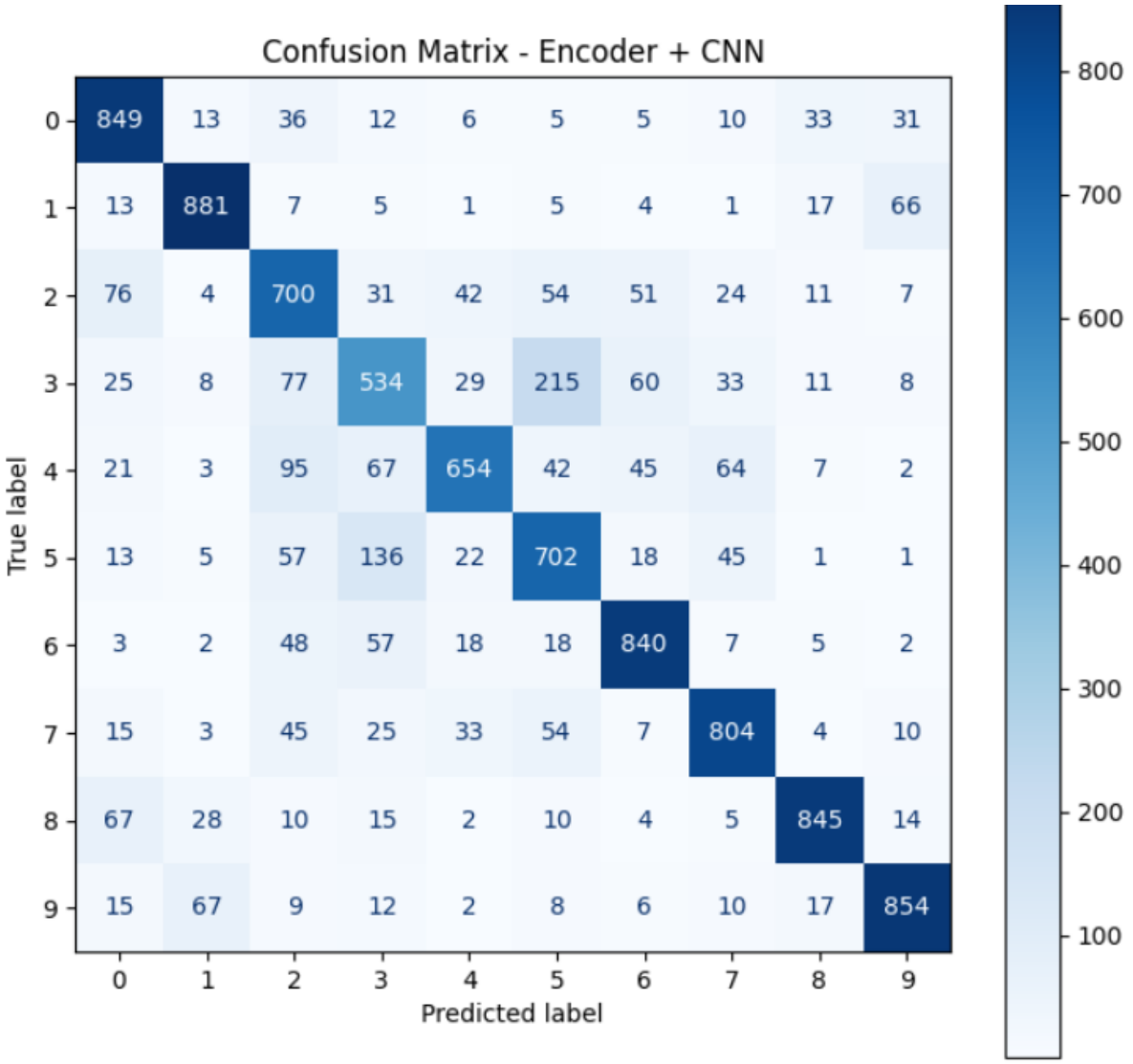

Confusion Matrix - Encoder + CNN
True label
Predicted label
0 661 27 82 29 16 3 12 7 118 45
1 12 756 3 17 4 2 15 2 46 143
2 52 4 521 93 142 59 66 31 17 15
3 14 11 76 538 75 111 92 33 21 29
4 19 6 71 101 623 17 73 59 28 3
5 12 8 67 306 76 426 47 43 7 8
6 9 5 32 85 54 14 765 14 10 12
7 15 7 26 56 129 68 20 641 10 28
8 36 48 12 23 6 1 9 2 835 28
9 29 121 8 26 5 4 13 11 44 739
0 1 2 3 4 5 6 7 8 9
800
700
600
500
400
300
200
100

8b

Μοντέλο	Encoder	val-acc	test-acc
Encoder + CNN	Trainable	0.7696	0.7663







9

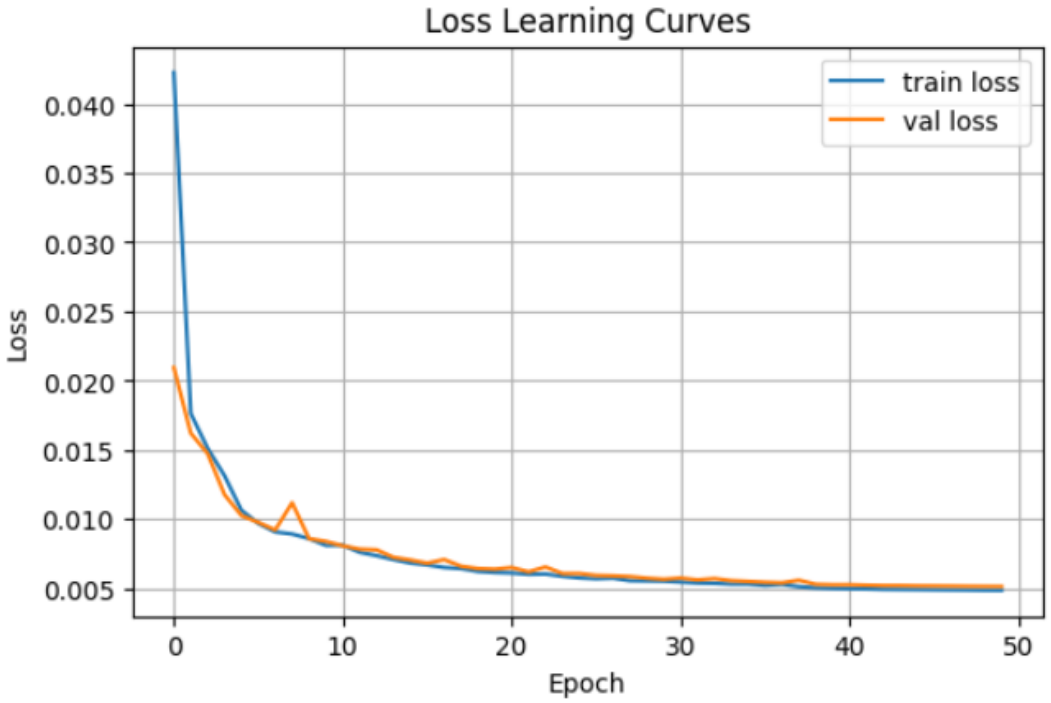
Το επομενο μοντελο ειναι Encoder + SVM for classification. Εδω, εφτιαξα νεο Encoder με latent space dimensions = 512, για να δινει διαστασεις στην εξοδο που μπορει να χειριστει το SVM. Χρησιμοποιησα LinearSVM ως το πιο γρηγορο.

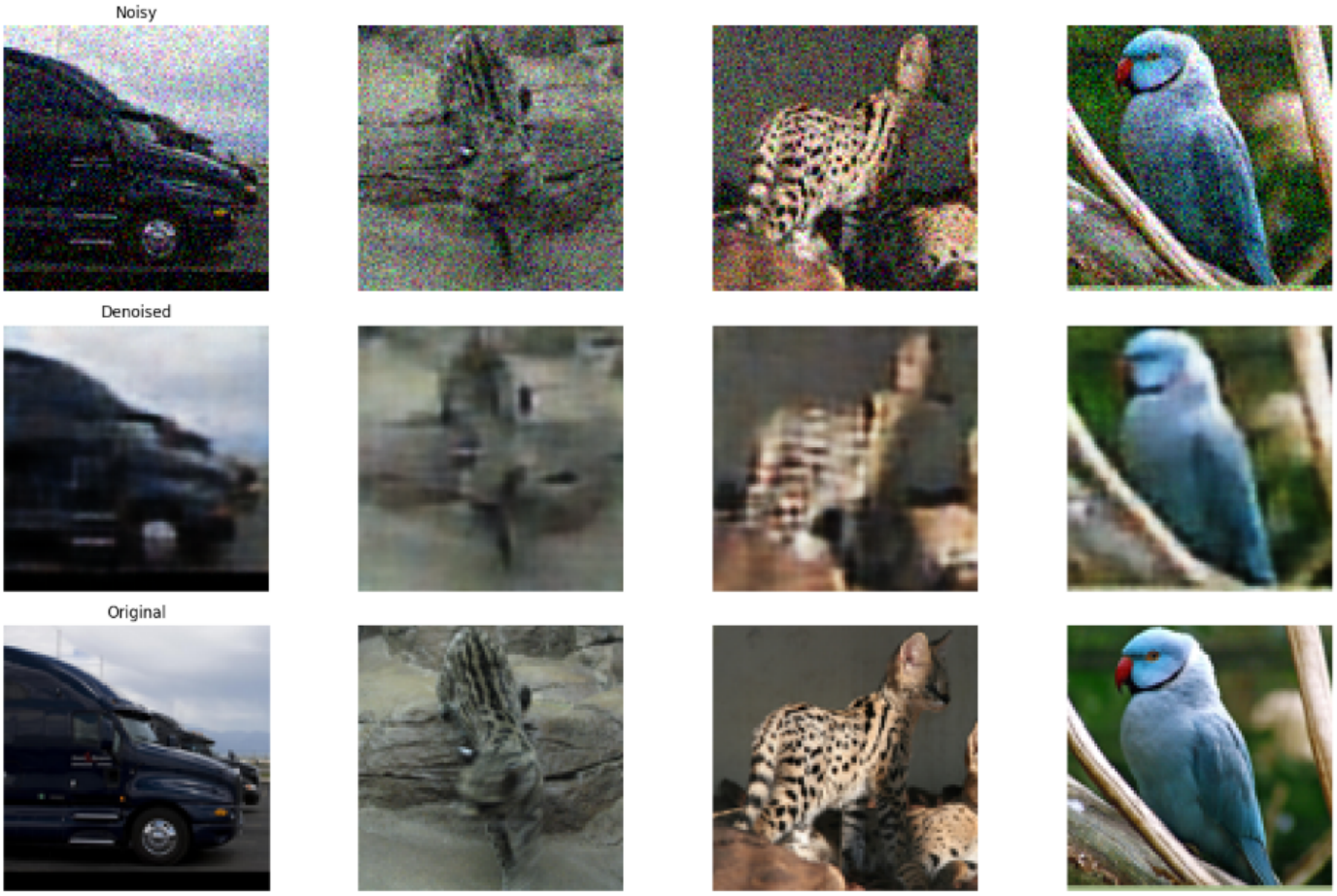
Μοντέλο	latent dim	val-acc	test-acc	Encoder time	SVM time
Encoder + SVM	512	0.5138	0.5121	305.51	47.21

10

Το τελευταιο πραγμα που εκανα στην εργασια ειναι ενας Denoising Autoencoder. Συγκεκριμενα, πηρα τις εικονες του STL-10 Dataset που ειναι διαστασεων (96,96,3), τους προσθεσα θορυβο τυπικης αποκλισης 0.1 και τις εδωσα στον Autoencoder με στοχο να τις ανακατασκευασει καθαρες. Οι latent space διαστασεις που χρησιμοποιησα ηταν 12x12x128 = 18.432 απο 96x96x3 = 27.648 που ειναι αρχικα η καθε εικονα. Τα αποτελεσματα φαινονται παρακατω:

Μοντέλο	latent dim	lr	val-loss	test-loss	time
Encoder + SVM	18.432	0.001	0.0052	0.0050	180.85




Noisy
Denoised
Original