# Νευρωνικά Δίκτυα - Βαθιά Μάθηση

Τμήμα Πληροφορικής ΑΠΘ 7ο εξάμηνο

## Ενδιάμεση Εργασία

**Αλεξία Νταντουρή** ΑΕΜ: 3871

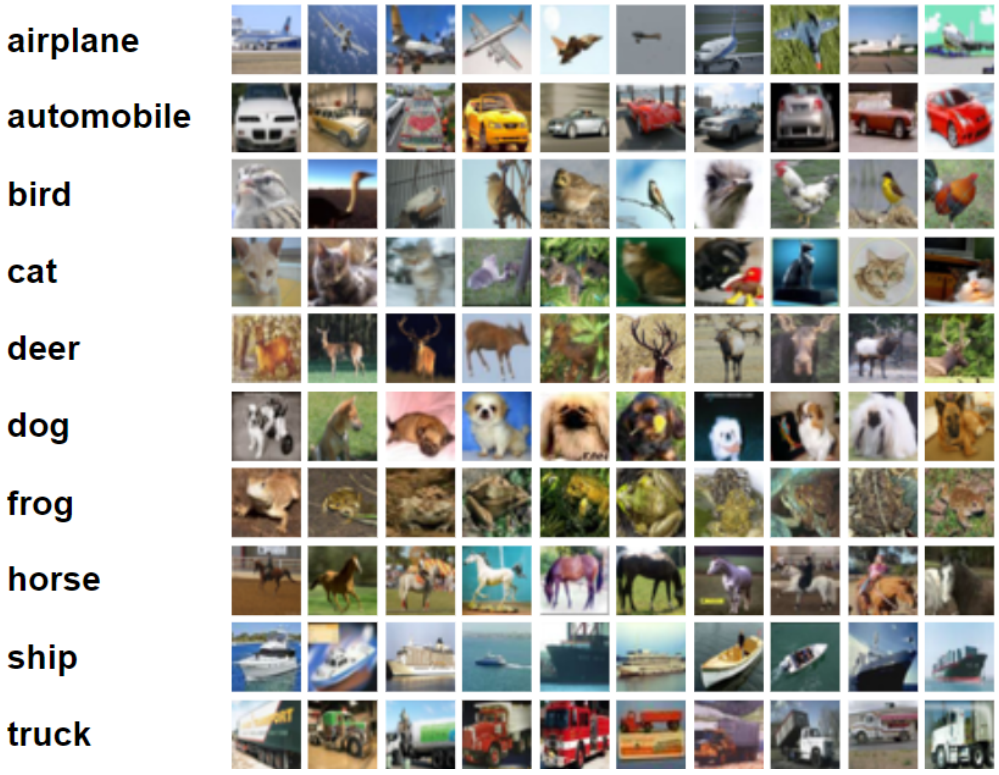
---

### Ενδιάμεση Εργασία

Να γραφεί πρόγραμμα σε οποιαδήποτε γλώσσα επιθυμείτε το οποίο να συγκρίνει την απόδοση του **κατηγοριοποιητή πλησιέστερου γείτονα με 1 και 3 πλησιέστερους γείτονες** με τον **κατηγοριοποιητή πλησιέστερου κέντρου** στην βάση δεδομένων που θα επιλέξετε για την εργασία σας.
Το πρόγραμμα δηλαδή αυτό θα πρέπει να διαβάζει τα δεδομένα εκπαίδευσης (training) και τα δεδομένα ελέγχου (test) και να μετράει την απόδοση των παραπάνω κατηγοριοποιητών.

### Dataset

Το σύνολο δεδομένων που χρησιμοποιήθηκε είναι το CIFAR-10, το οποίο περιέχει 60000 RGB εικόνες 32x32x3 pixels = 3072 pixels. Οι εικόνες κατηγοριοποιούνται σε 10 κλάσεις ανάλογα με το περιεχόμενό τους (αεροπλάνο, αυτοκίνητο, πουλί, γάτα, ελάφι, σκύλος, βάτραχος, άλογο, πλοίο, φορτηγό).



## Επεξεργασία dataset

Οι διαστάσεις των πινάκων ήταν οι εξής:

```
X_train dimensions: (50000, 32, 32, 3)
X_test dimensions: (10000, 32, 32, 3)
```

Και μετατράπηκαν σε:

```
X_train dimensions: (50000, 3072)
X_test dimensions: (10000, 3072)
```

ώστε κάθε εικόνα να αναπαρίσταται από ένα vector μεγέθους 3072.

Οι τιμές του vector έχουν κανονικοποιηθεί και κυμαίνονται στο διάστημα [0,1] ενώ αρχικά κυμαίνονταν στο διάστημα [0,255]. Κάθε στοιχείο του vector αναπαριστά ένα pixel μιας εικόνας.

Οι διαστάσεις του output set παρέμειναν οι εξής:

```
y_train dimensions: (50000, 1)
y_test dimensions: (10000, 1)
```

## K-Nearest Neighbors

Για τον κατηγοριοποιητή πλησιέστερων γειτόνων δημιουργήθηκε η συνάρτηση `myKNN_predictions`, η οποία παίρνει ως παραμέτρους, τα δεδομένα εκπαίδευσης (`X_train` και `y_train`), τα δεδομένα ελέγχου (`X_test`) και το πλήθος των γειτόνων (`k_neighbors`) και επιστρέφει έναν πίνακα με τις προβλέψεις για το σύνολο `X_test`.

Συγκεκριμένα, για κάθε δείγμα του test set υπολογίζονται οι ευκλείδειες αποστάσεις προς κάθε δείγμα του training set και αποθηκεύονται σε ένα πίνακα.
Στη συνέχεια, οι αποστάσεις ταξινομούνται και λαμβάνονται για κάθε δείγμα του test set οι δείκτες στηλών που δείχνουν στις k_neighbors μικρότερες αποστάσεις.
Έπειτα, από τους δείκτες, λαμβάνονται τα labels και με ψηφοφορία ανάμεσα στου k πλησιέστερους γείτονες γίνεται οι πρόβλεψη για κάθε δείγμα του test set.

```
def myKNN_predictions(X_train, y_train, X_test, k_neighbors):
  # compute the distances
  # each row has the euclidean distances of one sample from the test
  # set to all the samples from the training set
  distances = euclidean_distances(X_test, X_train)

  # sort the distances of each row and retrieve first k indices
  # these indices refer to the k nearest neighbors for each sample of
  # the test set
  indices = np.argsort(distances, axis=1)[:, :k_neighbors]

  # replace indices with labels
  for i in range(len(X_test)):
    for j in range(k_neighbors):
```

```
      indices[i][j] = y_train[indices[i][j]]

  # get the label with the most votes for each sample of the test set
  predictions = np.array([np.bincount(row).argmax() for row in
indices])

  # return predictions
  return predictions
```

### Nearest Centroid Classifier

Για τον κατηγοριοποιητή πλησιέστερου κέντρου δημιουργήθηκε η συνάρτηση `myNCC_predictions`, η οποία παίρνει ως παραμέτρους, τα δεδομένα εκπαίδευσης (`X_train` και `y_train`) και τα δεδομένα ελέγχου (`X_test`) και επιστρέφει έναν πίνακα με τις προβλέψεις για το σύνολο `X_test`.

Συγκεκριμένα, υπολογίζονται τα κέντρα χρησιμοποιώντας τη συνάρτηση `calculate_centroids` και στη συνέχεια, υπολογίζονται οι ευκλείδειες αποστάσεις των δειγμάτων του test set από τα κέντρα.

Στο τέλος, αποδίδεται σε κάθε δείγμα του test set το label που αντιστοιχεί στο κοντινότερο κέντρο.

```
def calculate_centroids(X_train, y_train):
  count = np.zeros(10)
  centroids = np.array([ [0.0]*3072 for i in range(10)])

  for i in range(len(y_train)):
    centroids[y_train[i]] += X_train[i]
    count[y_train[i]] += 1

  for i in range(10):
    centroids[i] /= count[i]

  return centroids

def myNCC_predictions(X_train, y_train, X_test):
  # calculate centroids
  centroids = calculate_centroids(X_train, y_train)

  # calculate the distances
  # each row has the euclidean distances of one sample
  # from the test set to all the centroids
  distances = euclidean_distances(X_test, centroids)

  # find the index of minimum distance of each row
```

```
  # the index refers to the nearest centroid for each sample of the
test set
  # predictions = np.argsort(distances, axis=1)[:, :1]
  predictions = np.argmin(distances, axis=1)

  return predictions
```

### Επαλήθευση

Χρησιμοποιήθηκαν οι συναρτήσεις `KNeighborsClassifier` και `NearestCentroid` της **sklearn** για επαλήθευση και είχαν το ίδιο accuracy με τις δικές μου συναρτήσεις. Επίσης, δεν είχαν μεγάλες αποκλίσεις όσον αφορά τον χρόνο.

Βέβαια, η συνάρτηση `myKNN_predictions` δεν είναι αποδοτική όσον αφορά την αξιοποίηση της RAM, καθώς δημιουργεί πίνακα ευκλείδιων αποστάσεων από κάθε δείγμα του test set προς όλα τα δείγματα του train set και στη συνέχεια τα ταξινομεί.

Έτσι, αν είχαμε ένα πολύ μεγάλο test set ή αν θέλαμε να υπολογίσουμε το train accuracy με την `myKNN_predictions`, θα είχαμε θέμα μνήμης.

### Αποδόσεις

Για το **test set**:

| | **my KNN** | |
|---|---|---|
| | **time (sec)** | **accuracy** |
| **1 neighbor** | 142.834 | 0.3539 |
| **3 neighbors** | 141.626 | 0.3303 |

| | **sklearn KNN** | |
|---|---|---|
| | **time (sec)** | **accuracy** |
| **1 neighbor** | 99.879 | 0.3539 |
| **3 neighbors** | 99.067 | 0.3303 |

| **my NCC** | |
|---|---|
| **time (sec)** | **accuracy** |
| 1.566 | 0.2774 |

| **sklearn NCC** | |
|---|---|
| **time (sec)** | **accuracy** |
| 0.732 | 0.2774 |

Παρατηρούμε ότι

- Ο αλγόριθμος **KNN με 1 γείτονα** είναι **λίγο πιο ακριβής** από τον **KNN με 3 γείτονες**.
- Ο αλγόριθμος **NCC** είναι **σημαντικά πιο γρήγορος** αλλά **λιγότερο ακριβής** από τον **KNN**.
- Ο αλγόριθμος **KNN** είναι **σημαντικά πιο αργός** αλλά **πιο ακριβής** από τον **NCC**.

Πιο συγκεκριμένα:
Από **KNN 1** σε **KNN 3,** έχουμε

- Μείωση ακρίβειας κατά **6.69%**

Από **KNN 1** σε **NCC,** έχουμε

- Μείωση χρόνου κατά **98.9%**
- Μείωση ακρίβειας κατά **21.62%**

Από **KNN 3** σε **NCC,** έχουμε

- Μείωση χρόνου κατά **98.89%**
- Μείωση ακρίβειας κατά **16.02%**

Για το **train set**:
(For KNN only 1 batch was used because `myKNN_predictions` is not scalable and requires too much RAM)

| | **my KNN** | |
|---|---|---|
| | **time (sec)** | **accuracy** |
| **1 neighbor** | 21.217 | 1.0 |
| **3 neighbors** | 18.416 | 0.543 |

| | **sklearn KNN** | |
|---|---|---|
| | **time (sec)** | **accuracy** |
| **1 neighbor** | 21.295 | 1.0 |
| **3 neighbors** | 20.068 | 0.543 |

| **my NCC** | |
|---|---|
| **time (sec)** | **accuracy** |
| 1.153 | 0.2697 |

| **sklearn NCC** | |
|---|---|
| **time (sec)** | **accuracy** |
| 1.163 | 0.2697 |

Όσον αφορά το train set, ο αλγόριθμος KNN με 1 γείτονα έχει train accuracy 1.0, καθώς ο 1ος πλησιέστερος γείτονας είναι το ίδιο το σημείο για το οποίο είναι γνωστό το label του.

Ο αλγόριθμος KNN με 3 γείτονες έχει έχει train accuracy 0.54 το οποίο είναι διπλάσιο από το 0.27 test accuracy (για 1 batch).

Τέλος, ο αλγόριθμος NCC έχει train accuracy 0.27 που είναι λίγο λιγότερα αλλά σχεδόν ίδιο με το 0.28 test accuracy.