Εργασία στα Νευρωνικά Δίκτυα

ΑΝΔΡΕΑ ΣΕΓΚΑΝΙ ΑΕΜ 10770

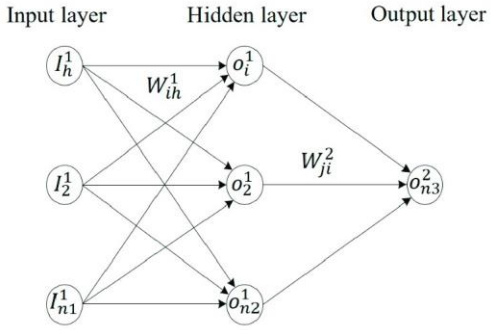
Η εργασία αυτή έχει ως στόχο την ανάπτυξη ενός νευρωνικού δικτύου εμπρόσθιας τροφοδότησης (Feedforward Neural Network), το οποίο υλοποιήθηκε σε Python.Το δίκτυο που επιλέχθηκε είναι πλήρως συνδεδεμένο (Multi-Layer Perceptron, MLP) και βασίζεται στον αλγόριθμο πίσω διάδοσης σφάλματος (backpropagation) για την εκπαίδευση του.

Το πρόβλημα που αντιμετωπίστηκε από το δίκτυο ήταν η κατηγοριοποίηση δεδομένων 10 κλάσεων με συνεχής εκπαίδευση. Τα δεδομένα που χρησιμοποιούνται προέρχονται από το σύνολο CIFAR-10 , το οποίο περιλαμβάνει εικόνες δέκα κατηγοριών, όπως ζώα και αντικείμενα.

Σκοπός της εργασίας είναι η επεξεργασία των δεδομένων που αποτελούν σημαντικά στάδια στη διαδικασία, όπως είναι η επεξεργασία των διαστάσεων των εικόνων , του αριθμού των hidden layers καθώς και των νευρώνων τους , και των τιμών του ρυθμού εκμάθησης.

Περιλαμβάνεται λεπτομερής ανάλυση της υλοποίησης, ενδεικτικά παραδείγματα σωστής και λανθασμένης κατηγοριοποίησης, καθώς και σύγκριση των αποτελεσμάτων με κλασικές μεθόδους, όπως ο πλησιέστερος γείτονας (Nearest Neighbor) και το πλησιέστερο κέντρο κλάσης (Nearest Class Centroid). Μέσα από τη συγκριτική αυτή προσέγγιση, αναδεικνύονται τόσο τα πλεονεκτήματα όσο και οι περιορισμοί της προτεινόμενης μεθόδου.

ΤΥΠΟΣ ΔΙΚΤΥΟΥ MLP



Στην παρούσα εργασία χρησιμοποιήθηκε το **Multilayer Perceptron (MLP)**, το οποίο είναι ένα πλήρως συνδεδεμένο νευρωνικό δίκτυο. Το MLP αποτελείται από τρία επίπεδα: το επίπεδο εισόδου, τουλάχιστον ένα κρυφό επίπεδο και το επίπεδο εξόδου. Κάθε νευρώνας του δικτύου είναι συνδεδεμένος με όλους τους νευρώνες του επόμενου επιπέδου, κάτι που του επιτρέπει να μάθει περίπλοκες μη γραμμικές σχέσεις στα δεδομένα.

A. Input Layer

Το επίπεδο εισόδου δέχεται τα δεδομένα της εικόνας, όπου κάθε εικόνα του CIFAR-10 έχει μέγεθος 32x32 pixels και 3 κανάλια χρωμάτων (RGB). Αυτό σημαίνει ότι κάθε είσοδος στο δίκτυο έχει 3072 χαρακτηριστικά (32x32x3). Κάθε χαρακτηριστικό αντιστοιχεί σε μία τιμή που περιγράφει τη φωτεινότητα του pixel για κάθε κανάλι χρώματος (R, G, B).

B. Hidden Layers

- Το πρώτο κρυφό επίπεδο που χρησιμοποιήθηκε στην εργασία αποτελείται από έναν αριθμό νευρώνων (τελικός αριθμός 256) , ο οποίος καθορίζεται από τον χρήστη κατά την εκκίνηση του δικτύου. Για τον μετασχηματισμό των εισερχόμενων δεδομένων, χρησιμοποιείται η συνάρτηση ενεργοποίησης **sigmoid**, η οποία εισάγει μη γραμμικότητα στο μοντέλο, επιτρέποντας στο δίκτυο να μάθει πιο σύνθετα μοτίβα.

- Στο δίκτυό χρησιμοποιήθηκαν δύο κρυφά επίπεδα (το πρώτο είχε 256 νευρώνες ενώ το πρώτο είχε 128 νευρώνες) για το οποίο οι έξοδοι του πρώτου κρυφού επιπέδου αποτελούν την είσοδο του.

C. Feedforward Algorithm

```
# Μέθοδος feedforward
def feedforward(self, X):
  # net activation (net = w * x)
  self.net1 = np.dot(X, self.weights_input_hidden1)
  # υπολογισμός από input σε hidden z = w * x + bias
  self.hidden_activation1 = self.net1 + self.bias_hidden1
  self.hidden1_output = self.sigmoid(self.hidden_activation1)
  #Από Layer 1 σε layer 2
  self.net2 = np.dot(self.hidden1_output, self.weights_hidden1_hidden2)
  self.hidden_activation2 = self.net2 + self.bias_hidden2
  self.hidden2_output = self.sigmoid(self.hidden_activation2)
  #apo layer 2 se output
  self.output_activation = np.dot(self.hidden2_output, self.weights_hidden2_output) + self.bias_ou
  self.predicted_output = self.sigmoid(self.output_activation)
  return self.predicted_output
```

Η διαδικασία feedforward αποτελεί το αρχικό στάδιο λειτουργίας ενός νευρωνικού δικτύου. Σε αυτό το στάδιο, τα δεδομένα εισόδου διατρέχουν διαδοχικά όλα τα επίπεδα του δικτύου, από την είσοδο έως την έξοδο, δηλαδή με μία και μόνο κατεύθυνση. Κατά τη διάρκεια αυτής της διαδικασίας, υπολογίζονται οι ενεργοποιήσεις των νευρώνων σε κάθε επίπεδο, με βάση τα βάρη των συνδέσεων (weights) και τις προκαταλήψεις (bias). Αναλυτικά, τα βήματα της διαδικασίας έχουν ως εξής:

- Τα δεδομένα εισόδου X πολλαπλασιάζονται με τα βάρη του πρώτου κρυφού επιπέδου W1.Κάθε είσοδος από το διάνυσμα εισόδου X περνά από τα επίπεδα του δικτύου, και η καθαρή ενεργοποίηση σε κάθε επίπεδο υπολογίζεται ως το εσωτερικό γινόμενο των εισόδων με τα βάρη του επιπέδου, προσθέτοντας τα bias. Επομένως, θα έχουμε,

$$\mathbf{net_j = \sum_{i=0}^{d} x_i * w_{ji}}$$

Τα δεδομένα εισόδου X πολλαπλασιάζονται με τα βάρη του πρώτου κρυφού επιπέδου

$$net_1 = x_1 * w_1$$

Στη συνέχεια, προστίθεται το bias του πρώτου επιπέδου b1 για κάθε νευρώνα:

$$\mathbf{z_1 = w_1 * x_1 + bias_1}$$

η έξοδος του πρώτου κρυφού επιπέδου προκύπτει με τη χρήση της συνάρτησης ενεργοποίησης:

$$\mathbf{z1\_out = sigmoid(z_1)}$$

και τέλος, χρησιμοποιείται για τον υπολογισμό της καθαρής ενεργοποίησης net_2 του δεύτερου κρυφού επιπέδου και αφού υπολογιστεί η έξοδος του κρυφού επιπέδου(αφού προστεθεί και το bias) ,προκύπτει ότι η καθαρή ενεργοποίηση της εξόδου είναι :

$$\mathbf{net\_out = w_2 * z2\_out + bias_{output}}$$

Τελικά , η έξοδος του δικτύου είναι:

$$\mathbf{output = sigmoid(net\_out)}$$

D. Backpropagation Algorithm

```
# Υπολογισμός του backpropagation
def back(self, X, y):

  # Error output
  delta_k = (y - self.predicted_output) * self.sigmoid_derivative(self.predicted_output) # δk = (tk - zk) * f'(netk)

  #Error hidden layer 2
  delta_j2 = self.sigmoid_derivative(self.z2_out) * np.dot(delta_k, self.weights_z2_out.T)
  #Error hidden layer 1
  delta_j1 = self.sigmoid_derivative(self.z1_out) * np.dot(delta_j2, self.weights_hidden1_hidden2.T)



  # Υπολογισμός των weights και bias για το output
  self.weights_z2_out += np.dot(self.z2_out.T, delta_k) * self.learning_rule
  self.bias_output += np.sum(delta_k, axis=0, keepdims=True) * self.learning_rule # bk = η * Σ(δk)
  # Υπολογισμός των weights και bias για το hidden layer 2
  self.weights_hidden1_hidden2 += np.dot(self.z1_out.T, delta_j2) * self.learning_rule
  self.bias_hidden2 += np.sum(delta_j2, axis=0, keepdims=True) * self.learning_rule # bj = η * Σ(δj)
  # Υπολογισμός των weights και bias για το hidden layer 1
  self.weights_input_hidden1 += np.dot(X.T, delta_j1) * self.learning_rule
  self.bias_hidden1 += np.sum(delta_j1, axis=0, keepdims=True) * self.learning_rule # bj = η * Σ(δj)
```

Στη μέθοδο backpropagation ,στη διαδικασία εκπαίδευσης, τα βάρη μεταξύ του κρυφού και του εξόδου υπολογίζονται με βάση το σφάλμα που προκύπτει από τη διαφορά μεταξύ της πραγματικής και της επιθυμητής εξόδου.
ο σφάλμα της εξόδου δ_κ υπολογίζεται ως η διαφορά μεταξύ της πραγματικής τιμής t_k και της υπολογισμένης εξόδου σήματος που εκπέμπει το δίκτυο z_k, πολλαπλασιασμένη με τη παράγωγο της συνάρτησης ενεργοποίησης στο επίπεδο εξόδου:

$$\mathbf{\delta_k = (t_k - z_k) * f'(net_k)}$$

Εδώ, f′(net$_k$) είναι η παράγωγος της συνάρτησης ενεργοποίησης sigmoid που εφαρμόζεται στις ενεργοποιήσεις του δικτύου. Το σφάλμα στο δεύτερο κρυφό επίπεδο (δ_{j2}) υπολογίζεται μεταφέροντας το σφάλμα από το επίπεδο εξόδου πίσω, μέσω των βαρών που συνδέουν το επίπεδο εξόδου με το δεύτερο κρυφό επίπεδο. Το αποτέλεσμα πολλαπλασιάζεται με την παράγωγο της sigmoid στο net_2 (ενεργοποίηση του δεύτερου κρυφού επιπέδου) σύμφωνα με τον τύπο:

$$\delta_j \equiv f'(net_j) \sum_{k=1}^{c} w_{kj}\delta_k. \quad (1)$$

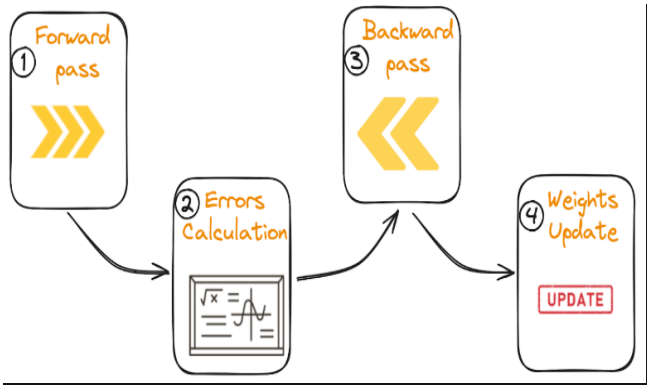
Όπου **f'(net$_2$) = f'(z$_{2out}$)**.

Ομοίως, το σφάλμα στο πρώτο κρυφό επίπεδο υπολογίζεται (σύμφωνα με την (1)) μεταφέροντας το σφάλμα από το δεύτερο κρυφό επίπεδο πίσω, μέσω των βαρών που συνδέουν το πρώτο και το δεύτερο κρυφό επίπεδο. Σε κάθε επίπεδο ενημερώνονται και τα βάρη καθώς και τα bias σύμφωνα με τους κανόνες:

$$\Delta w_{kj} = \eta \delta_k y_j \text{ (για τα βάρη)}$$

$\Delta b_k = \eta^* \sum \delta_k$ (για τα bias)

Η διαδικασία backpropagation διασφαλίζει ότι τα βάρη του δικτύου προσαρμόζονται σε κάθε επανάληψη εκπαίδευσης, μειώνοντας το σφάλμα της εξόδου. Η ανάλυση που έγινε μπορεί να φανεί αυτούσια στον κώδικα.



ΔΙΑΔΙΚΑΣΙΑ ΕΚΠΑΙΔΕΥΣΗΣ

Αρχικά, αφού το dataset αποθηκεύτηκε στο Google Drive και δόθηκε το κατάλληλο path για την πρόσβαση στο dataset, τα δεδομένα εισάγονται στον κώδικα για επεξεργασία. Στη συνέχεια, εφαρμόζεται ένας **μετασχηματισμός (transform)** στα δεδομένα μέσω της βιβλιοθήκης **Py**Torch για να τα προετοιμάσουμε κατάλληλα για την εκπαίδευση του μοντέλου.

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

Η χρήση του transform εξασφαλίζει ότι τα δεδομένα:

- Είναι στη σωστή μορφή για να περάσουν στο νευρωνικό δίκτυο.
- Είναι κανονικοποιημένα, ώστε να επιταχύνεται η εκπαίδευση και να αποφεύγονται αριθμητικά προβλήματα.
- Είναι κατάλληλα διαχειρίσιμα από το DataLoader, κάτι που βελτιώνει τη συνολική ροή εκπαίδευσης.
-

Έπειτα το δίκτυο «διαβάζει» τα δεδομένα από το test και train loaders χωρίζοντας τα σε 64 batches το καθένα , το οποίο θεωρείται η πιο σταθερή επιλογή.

Στο δίκτυο που κατασκευάστηκε χρησιμοποιήθηκαν **δύο hidden layers** , αφού στη περίπτωση του ενός hidden layer είχαμε επίδοση εκπαίδευσης 27% το οποίο είναι αρκετά χαμηλότερο από τη περίπτωση των 2 layers όπου ,τελικά, έχουμε training accuracy περίπου 75-80% και test accuracy περίπου 45% , όπως θα δούμε για διάφορες παραμέτρους κάτω. Σε αυτά τα αποτελέσματα φτάσαμε με τις εξής παραμέτρους :

```
input_size = 32*32*3
hidden1_size = 256
hidden2_size = 128
output_size = 10
learning_rate = 0.01
epochs = 60
```

Ο αλγόριθμος του train έτρεξε για 34 λεπτά ενώ οι επιλογές αυτές είναι οι βέλτιστες για το δίκτυο μας.

Ο training αλγόριθμος του mlp που χρησιμοποιήθηκε κατέληξε να είναι ο εξής:

```
# Μέθοδος εκπαίδευσης
def train(self, X, y, epochs):
  for epoch in range(epochs):
    output = self.feedforward(X)
    self.back(X, y)
    loss = np.mean(np.square(y - output))
```

Η συνάρτηση train εκπαιδεύει το νευρωνικό δίκτυο για τον αριθμό των εποχών που καθορίζεται ως είσοδος. Σε κάθε εποχή, καλείται πρώτα η μέθοδος feedforward για να υπολογιστούν οι προβλέψεις του δικτύου με βάση τα δεδομένα εισόδου, και στη συνέχεια η μέθοδος back εκτελεί backpropagation για την προσαρμογή των βαρών του δικτύου. Η συνάρτηση np.mean(np.square(y - output)) υπολογίζει την μέση τετραγωνική απώλεια. Μεγαλύτερη απόδοση θα έχουμε όταν η μέση τιμή της διαφοράς των τετραγώνων πραγματικής ετικέτας y και εκείνης της πρόβλεψης του νευρωνικού, output γίνει μικρότερη.

Για να εφαρμοστεί ο αλγόριθμος έπρεπε οι εικόνες να μετατραπούν σε vectors σύμφωνα με τον εξής κώδικα:

X_batch = X_batch.numpy().reshape(X_batch.shape[0], -1)

Τα MLP μοντέλα δουλεύουν μόνο με μονοδιάστατες εισόδους, οπότε πρέπει να "ισοπεδώσουμε" (flatten) τις εικόνες. Αυτό σημαίνει ότι κάθε pixel της εικόνας γίνεται μία τιμή σε ένα διάνυσμα εισόδου.

TRAINING

```
for epoch in range(epochs):
    total_correct = 0
    total_samples = 0

    for X_batch, y_batch in training_loader:
        # Flatten τα δεδομένα
        X_batch = X_batch.numpy().reshape(X_batch.shape[0], -1)
        y_batch = y_batch.numpy()
        #one-hot encoding
        y_batch_onehot = np.zeros((y_batch.size, output_size))
        y_batch_onehot[np.arange(y_batch.size), y_batch.flatten()] = 1

        mlp.feedforward(X_batch)
        mlp.back(X_batch, y_batch_onehot)
        outputs = mlp.predicted_output
        predictions = np.argmax(outputs, axis=1)
        total_correct += (predictions == y_batch.flatten()).sum()
        total_samples += y_batch.size

    #accuracy
    train_accuracy = 100 * total_correct / total_samples
```

1. Τα δεδομένα χωρίζονται σε μικρότερα σύνολα (batches) για να επεξεργάζονται αποδοτικά. Κάθε batch περιέχει:

- X_batch: Τα χαρακτηριστικά εισόδου του batch.
- y_batch: Οι ετικέτες (labels) των δειγμάτων του batch

2. Τα δεδομένα εισόδου από τις εικόνες μετατρέπονται σε διανύσματα (flatten) για να περάσουν στο MLP

3. Μετατρέπουμε τα labels σε one-hot encoded διανύσματα. Αυτό γιατί δεν γινόταν να υπολογισθεί η αφαίρεση πινάκων μεταξύ y και self.predicted_output στον αλγόριθμο του mlp. Αυτό συμβαίνει διότι το y είναι ένας μονοδιάστατος πίνακας (batch_size,) που περιέχει ακέραιες τιμές (labels) για κάθε δείγμα στο batch , ενώ το self.predicted_output είναι δισδιάστατος πίνακας με μέγεθος (64,10) για κάθε μία από τις 10 κατηγορίες. Έτσι βρήκα τη one-coded μορφή ώστε το y να είναι και αυτό (64,10)

4. Εφαρμόζουμε τον αλγόριθμο του mlp με τις μεθόδους feedforward και backpropagation

5. Ελέγχεται πόσες προβλέψεις του μοντέλου είναι σωστές για το batch, και ενημερώνονται οι συνολικές μετρήσεις

6. Μετά την επεξεργασία όλων των batches, η ακρίβεια υπολογίζεται ως ποσοστό των σωστών προβλέψεων στο σύνολο των δειγμάτων

TESTING

```
for X_batch, y_batch in test_loader:
    # Flatten τα δεδομένα
    X_batch = X_batch.numpy().reshape(X_batch.shape[0], -1)
    y_batch = y_batch.numpy()

    # Feedforward μόνο
    outputs = mlp.feedforward(X_batch)

    # Προβλέψεις
    predictions = np.argmax(outputs, axis=1)
    total_correct += (predictions == y_batch.flatten()).sum()
    total_samples += y_batch.size

# Υπολογισμός Test Accuracy
test_accuracy = 100 * total_correct / total_samples
```

Όπως φαίνεται και στον testing αλγόριθμο της εικόνας,

1. Το test set αφού γίνει το flatten περνά περνάει από το νευρωνικό σε batches

2. Υπολογίζονται οι προβλέψεις (outputs) με τη μέθοδο feedforward. Αυτές είναι οι πιθανότητες για κάθε κατηγορία

3. Βρίσκεται η κατηγορία με τη μεγαλύτερη πιθανότητα από τα outputs για κάθε δείγμα με την εντολή np.argmax. Οι προβλέψεις predictions περιέχουν τους δείκτες των κατηγοριών που το μοντέλο πιστεύει ότι είναι σωστές

4. Γίνεται σύγκριση μεταξύ των προβλέψεων predictions και των πραγματικών ετικετών y_batch. Το total_correct βρίσκει τον αριθμό των σωστών προβλέψεων όταν το prediction ταυτίζεται με το y , αυξάνοντας τον αριθμό του test_sample για να γνωρίζουμε τι έχει δοκιμαστεί μέχρι τώρα

5. Βρίσκεται το test accuracy σε ποσοστό

Αποτελέσματα

Μετά από πολλές δοκιμές κατέληξα ότι έχουμε τη πιο βέλτιστη απόδοση στα 40-45 epochs . Για τις δοκιμές θα χρησιμοποιήσω 40 epochs .
256 νευρώνες στο πρώτο layer και 128 νευρώνες στο δεύτερο layer με learning rate 0.01 έχουμε τα εξής αποτελέσματα.

```
Epoch 45/45, Training Accuracy: 80.83000%
Test Accuracy: 45.17%
```

40 εποχές :

```
Epoch 40/40, Training Accuracy: 78.78200%
Test Accuracy: 45.35%
```

Στη συνέχεια για learning rate 0.03 έχουμε :

```
 Epoch 40/40, Training Accuracy: 65.97000%
Test Accuracy: 46.18%
```

Το test accuracy είναι υψηλότερο (που είναι ο κύριος στόχος) και η διαφορά μεταξύ train και test accuracy είναι μικρότερη, κάτι που δείχνει καλύτερη ισορροπία μεταξύ εκπαίδευσης και γενίκευσης.

Στη συνέχεια για learning rate 0.045 έχουμε :

```
Epoch 40/40, Training Accuracy: 58.87600%
Test Accuracy: 45.07%
```

Βλέπουμε πως έχουν πέσει οι αποδόσεις , λογικά

Εδώ έχουμε l =0.007

```
Epoch 40/40, Training Accuracy: 81.26400%
Test Accuracy: 46.04%
```

Εδώ ίσως με περισσότερα epochs να είχαμε λίγο καλύτερη απόδοση

Ίδιος αριθμός νευρώνων (128) με l=0.03

```
Epoch 40/40, Training Accuracy: 64.82800%
Test Accuracy: 46.48%
```

Ίδιος αριθμός νευρώνων (256) στα δύο επίπεδα (l=0.03)

```
Epoch 40/40, Training Accuracy: 24.43400%
Test Accuracy: 20.98%
```

Πολύ μεγάλος αριθμός νευρώνων σε κάθε επίπεδο μπορεί να κάνει το δίκτυο να χρειάζεται περισσότερες εποχές για να συγκλίνει. Οπότε θεωρητικά, θα θέλουμε >=100 εποχές. Επίσης από βιβλιογραφία βρήκα ότι θα μπορούσαμε να βάλουμε κάποιου είδους regulator στο output, απλά δεν είχα χρόνο να μπορέσω να το κάνω έγκαιρα για να το υποβάλλω. Γενικά, όμως, ο αριθμός νευρώνων μειώνεται προοδευτικά καθώς προχωράμε σε βαθύτερα επίπεδα.Αυτό γιατί κάθε επίπεδο εξάγει επιπέδου χαρακτηριστικά, οπότε απαιτούνται λιγότεροι νευρώνες για την αναπαράσταση αυτών των χαρακτηριστικών.

Τέλος για 32 batches (αντί για 64 που χρησιμοποιήθηκε) φτάσαμε σε `Test Accuracy: 46.42%` με `Training Accuracy: 66.75000%`. Αυτό υποδεικνύει ότι το μοντέλο γενικεύει καλά στα νέα, άγνωστα δεδομένα. Επειδή train και set βρίσκονται κοντά μεταξύ τους

Για το running time του αλγορίθμου , κάνοντας χρήση της περιορισμένης GPU του Google Collab χρειάστηκαν περίπου 30 λεπτά για το training αλλά αυτό αυξάνεται με τον αριθμό των εποχών , των νευρώνων ή και του learning rate , καθώς και με τη χρήση CPU αντί της GPU μιας και που είναι περιορισμένη η χρήση της.

Το total_correct δείχνει ακριβή αριθμό σωστού testing.

Στην ενδιάμεση εργασία είχε βρεθεί ότι η απόδοση KNN ήταν περίπου **30%** ενώ η απόδοση του nearest centroid ήταν **23%**
Οι χαμηλότερες αποδόσεις των KNN και Nearest Centroid πιθανότατα οφείλονται σε μη κανονικοποιημένα δεδομένα, μεγάλη διάσταση ή μη γραμμικές σχέσεις. Αυτοί οι αλγόριθμοι είναι πιο κατάλληλοι για μικρά datasets με λίγες διαστάσεις και γραμμικά διαχωρίσιμες κλάσεις.Επίσης εμείς χρησιμοποιήσαμε k=1 ,3 πράγμα το οποίο σημαίνει ότι κάνει overfitting αφού δεν μπορεί να γενικεύσει δεδομένα, εξού και τα χαμηλά ποσοστά.

Βιβλιογραφία

- https://medium.com/eincode/mastering-the-multi-layer-perceptron-mlp-for-image-classification-a0272baf1e29
- Chat GPT
- Διαφάνειες E-learning από όπου εμπνεύστηκα τον αλγόριθμο
- https://www.geeksforgeeks.org/ml-one-hot-encoding/
- https://stackoverflow.com/questions/66851759/knn-image-classification-bad-accuracy
- YouTube
- Stack Overflow