Νευρωνικά Δίκτυα
1η Εργασία

Αποστολίδου Αθηνά, ΗΜΜΥ, ΑΕΜ:10400

Περίληψη—Σε αυτό το έγγραφο παρουσιάζονται τα αποτελέσματα απο δυο αρχιτεκτονικές νευρωνικών δικτύων και συγκρίνονται με τους κατηγοριοποιήτες πλησιέστερου γείτονα με 1 και 3 γείτονες και του κατηγοριοποιητή πλησιέστερου κέντρου για την επίλυση της βάσης δεδομένων **Cifar-10**.

I . Εισαγωγή

ΣΤΟ πλαίσιο εργασίας του μαθήματος Νευρωνικά δίκτυα και βαθία μάθηση υλοποιήθηκαν δυο αρχιτεκτονικές νευρωνικών δικτύων, ενός πλήρως συνδεδεμένου και ενός με συνελικτικά layer, με σκοπό την επίλυση της βάσης Cifar-10. Η απόδοση των δύο αυτών νευρωνικών δικτύων συγκρίνεται με την απόδοση των κατηγοριοποιητών πλησιέστερου γείτονα και πλησιέστερους κέντρου. Για την εργασία χρησιμοποιήθηκε γλώσσα προγραμματισμού Python και βιβλιοθήκες numpy, scipy, pickle, time, sklearn.metrics, matplotlib, pytorch.

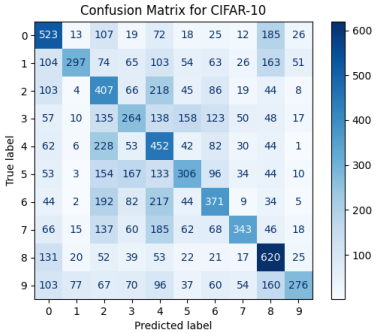
II . Κατηγοριοποιήτης Πλησιέστερου Γείτονα

Σε αυτή την ενότητα θα παρουσιαστούν τα αποτελέσματα του κατηγοροιοποιήτη πλησιέστερου γείτονα. Η λογική αυτού του κατηγοροιοποιητή είναι ότι εξετάζει τις αποστάσεις κάθε εικόνας στο test set με όλες τις εικόνες του training set και αποδίδει σε αυτή το label που κυριαρχεί ανάμεσα στις K κοντινότερες εικόνες. Η απόσταση που χρησιμοποιηθηκε είναι η Manhattan distance η οποία δίνεται απο την σχέση

$$d(p,q) = \sum_{k=1}^{n} |p_k - q_k|$$

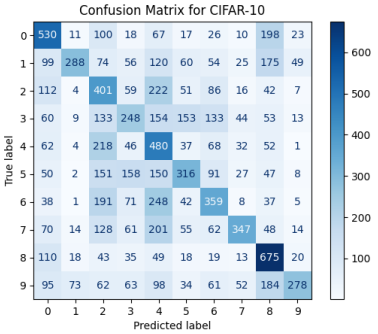
Στην παρούσα εργασία εξετάστηκαν οι περιπτώσεις $k = 1$ και $k = 3$

1) Η περίπτωση του 1ος γείτονα: Ο αλγόριθμος είχε χρόνο εκτέλεσης 1175 δευτερόλεπτα και πέτυχε ακρίβεια 38,59%. Τα αποτελέσματα φαίνονται στο παρακάτω confusion diagram.



Σχήμα 1: Η περίπτωση του ενός γείτονα

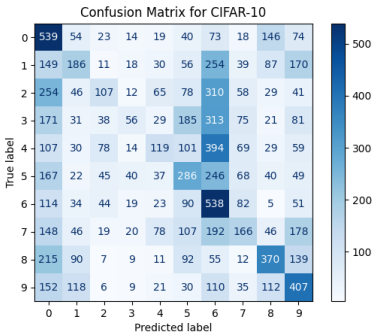
2) Η περίπτωση των 3ων πλησιέστερων γειτόνων: Ο αλγόριθμος είχε χρόνο εκτέλεσης 1100 δευτερόλεπτα και πέτυχε ακρίβεια 39,22%. Τα αποτελέσματα φαίνονται στο παρακάτω confusion diagram.



Σχήμα 2: Η περίπτωση των τριών γειτόνων

III . Κατηγοριοποιητής Πλησιέστερου Κέντρου

Στον κατηγοριοποιητή πλησιέστερου κέντρου αρχικά υπολογίζεται το κέντρο κάθε κλάσης του training set. Έπειτα εξετάζεται κάθε δείγμα του test set και παίρνει το label της κλάσης της οποίας το κέντρο είναι πιο κοντά σε αυτό. Η απόσταση που αναφέρεται και σε αυτή την περίπτωση είναι η Manhattan distance. Ο χρόνος εκτέλεσης του αλγορίθμου αυτού ήταν 0.43 δευτερόλεπτα και η ακρίβεια που πέτυχε 27,74%. Τα αποτελέσματα φαινονταί στο confusion diagram του σχήματος 3.



Σχήμα 3: Η περίπτωση του πλησιέστερου κέντρου

IV . Πλήρως συνδεδεμένο Νευρωνικό Δίκτυο

Το πλήρες συνδεδεμένο νευρωνικό δίκτυο κατασκευάστηκε χωρίς την χρήση ειδικών βιβλιοθηκών.

Α΄. Χαρακτηριστικά *MPL*

Το μέγεθος της εισόδου είναι $32 \cdot 32 \cdot 3$ λόγω του ότι και η οριζόντια και η κατακόρυφη διάσταση της είκονας είναι 32 pixel και υπάρχουν τα τρια κανάλια RGB. Επίσης, το νεωρωνικό θα έχει ένα κρυφό επίπεδο και το επίπεδο εξόδου θα έχει 10 νεωρώνες όσοι και το πλήθος των διαφορετικών κλάσεων των εικόνων της βάσης. Ως συνάρτηση ενεργοποίησης για το κρυφό επίπεδο χρησιμοποιήθηκε σε κάποια παραδείγματα η σιγμοειδής συνάρτηση όπως φαίνεται παρακάτω

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Ενώ για το φανερό επίπεδο χρησιμοποιήθηκε η συνάρτηση softmax

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

ενώ σε άλλα η συνάρτηση relu

$$\text{relu}(x) = \max(0, x)$$

Ως συνάρτηση απώλειας χρησιμοποιήθηκε η συνάρτηση cross entropy ο τύπος της οποίας δίνεται παρακάτω:

$$L = -\sum_{i=1}^{N} \sum_{j=1}^{C} y_{ij} \log(\hat{y}_{ij})$$

Όπου:

- N: ο αριθμός των δειγμάτων.
- C: ο αριθμός των κλάσεων.
- y_{ij}: το πραγματικό ετικετοποιημένο διάνυσμα (ονε-ηοτ) για το δείγμα i στην κλάση j.
- $\hat{y}_{ij}$: η πιθανότητα που προβλέπεται από το μοντέλο για το δείγμα i στην κλάση j (μετά από softmax).

Παρακάτω παρατίθεται ο κώδικας πλήρως σχολιασμένος

```python
# Η βιβλιοθήκη torchvision χρησιμοποιήθηκε μόνο για την φόρτωση του data set
import numpy as np
import torch
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import time
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay


transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=len(trainset), shuffle=True)
testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=len(testset), shuffle=False)

# Load the entire training set into memory
for data in trainloader:
    X_train, y_train = data
    break
# Load the entire test set into memory
for data in testloader:
    X_test, y_test = data
    break
```

Σχήμα 4: Φόρτωση βιβλιοθηκών και data set

```python
# Flatten the images and convert to NumPy arrays
X_train = X_train.view(X_train.size(0), -1).numpy()
y_train = y_train.numpy()
X_test = X_test.view(X_test.size(0), -1).numpy()
y_test = y_test.numpy()

# Αντιστοιχίζει τα labels σε έναν αριθμό
def one_hot_encode(y, num_classes):
    return np.eye(num_classes)[y]

y_train_one_hot = one_hot_encode(y_train, 10)
y_test_one_hot = one_hot_encode(y_test, 10)

#Η συνάρτηση relu
def relu(x):
    return np.maximum(0, x)
#Η παράγωγος της relu
def relu_gradient(x):
    return (x > 0).astype(float)


#Η sigmoid
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
#Η παράγωγος της sigmoid
def sigmoid_gradient(x):
    sig = sigmoid(x)
    return sig * (1 - sig)
```

Σχήμα 5: Ορισμός συναρτήσεων

```python
#Η συνάρτηση softmax
def softmax(x):
    exp_x = np.exp(x - np.max(x, axis=1, keepdims=True))  # For numerical stability
    return exp_x / np.sum(exp_x, axis=1, keepdims=True)


# Ορίζω το νεωρωνικό δίκτυο ως κλάση
class FullyConnectedNN:
    def __init__(self, input_size, hidden_size, output_size, learning_rate=0.04):
        np.random.seed(42)
        self.W1 = np.random.randn(input_size, hidden_size) * 0.01 #Ο πίνακας με τα βάρη ανάμεσα στην είσοδο και το κρυφό layer
        self.b1 = np.zeros((1, hidden_size))  # Το αντίστοιχο bias
        self.W2 = np.random.randn(hidden_size, output_size) * 0.01 #Ο πίνακας με τα βάρη ανάμεσα στο κρυφό layer και την έξοδο
        self.b2 = np.zeros((1, output_size))  # Το αντίστοιχο bias
        self.learning_rate = learning_rate #Το learning rate
        self.train_accuracies = []  # Αποθηκεύω training accuracies
        self.test_accuracies = []   # Αποθηκεύω test accuracies
        self.train_losses = []      # Αποθηκεύω training losses
        self.test_losses = []       # Αποθηκεύω test losses

    def forward(self, X):

        #Εμπρόσθιο πέρασμα της εισόδου στο νευρωνικό
        self.Z1 = np.dot(X, self.W1) + self.b1
        self.A1 = relu(self.Z1) #Για ενδιάμεση συνάρτηση ενεργοποίησης είτε relu είτε sigmoid
        self.Z2 = np.dot(self.A1, self.W2) + self.b2
        self.A2 = softmax(self.Z2)
        return self.A2
```

Σχήμα 6: Constructor και forward pass

```python
def backward(self, X, y_true):
    m = X.shape[0]
    y_pred = self.A2

    # Υπολογισμός error στο εξωτερικό layer
    dZ2 = y_pred - y_true
    dW2 = np.dot(self.A1.T, dZ2) / m
    db2 = np.sum(dZ2, axis=0, keepdims=True) / m

    # Υπολογισμός error στο hidden layer
    dA1 = np.dot(dZ2, self.W2.T)
    dZ1 = dA1 * relu_gradient(self.A1)
    dW1 = np.dot(X.T, dZ1) / m
    db1 = np.sum(dZ1, axis=0, keepdims=True) / m

    # Ανανέωση βαρών και biases
    self.W2 -= self.learning_rate * dW2
    self.b2 -= self.learning_rate * db2
    self.W1 -= self.learning_rate * dW1
    self.b1 -= self.learning_rate * db1

def compute_loss(self, y_true, y_pred):#Συνάρτηση απώλειας cross entropy
    return -np.mean(np.sum(y_true * np.log(y_pred + 1e-8), axis=1))  #Προσθέτω στο τέλος μια μικρή σταθερά
```

Σχήμα 7: Backprobagation and cross entropy loss

```python
def train(self, X, y, epochs=100, decay_factor=0.95):
    start_time = time.time() #Ξεκινάει η μετρητής του χρόνου εκπαίδευσης
    for epoch in range(epochs):
        if epoch % 20 == 0:
            self.learning_rate *= decay_factor #Κάθε 20 εποχές μειώνεται το learning rate
            # Την άλλη σταθερά learning rate βάζω decay_factor=1
        # Forward pass
        y_pred = self.forward(X)
        # Υπολογισμός training loss
        train_loss = self.compute_loss(y, y_pred)
        self.train_losses.append(train_loss)

        # Backward pass
        self.backward(X, y)

        # Υπολογισμός training accuracy
        train_accuracy = self.accuracy(X, np.argmax(y, axis=1))
        self.train_accuracies.append(train_accuracy)

        # Υπολογισμός test loss και accuracy
        y_test_pred = self.forward(X_test)  # Reuse forward pass for the test set
        test_loss = self.compute_loss(y_test_one_hot, y_test_pred)
        self.test_losses.append(test_loss)
        test_accuracy = self.accuracy(X_test, y_test)
        self.test_accuracies.append(test_accuracy)

        if epoch % 10 == 0:
            print(f'Epoch {epoch}, Loss: {train_loss:.4f}, Test Loss: {test_loss:.4f}, Train Accuracy: {train_accuracy:.4f}, Test Accuracy: {test_accuracy:.4f}')
    end_time = time.time()#ΤΕΛΟΣ χρονομέτρησης
    self.training_time = end_time - start_time  # Χρόνος εκπαίδευσης
```

Σχήμα 8: Training

```python
    def predict(self, X):
        # Πρόβλεψη κλάσης
        y_pred = self.forward(X)
        return np.argmax(y_pred, axis=1)  # Επιλέγει την κλάση με τη μεγαλύτερη πιθανότητα

    def accuracy(self, X, y):
        #Υπολογίζει την ακρίβεια
        y_pred = self.predict(X)
        correct = np.sum(y_pred == y)
        return correct / len(y)


input_size = 3072  # 32*32*3 για CIFAR-10
hidden_size = 1024 #Ελεύθερη επιλογή
output_size = 10 #Αφού έχουμε 10 εξόδους

nn = FullyConnectedNN(input_size, hidden_size, output_size)

# Κανονικοποίηση X_train και X_test σε διάστημα [0, 1]
X_train = (X_train - X_train.min()) / (X_train.max() - X_train.min())
X_test = (X_test - X_test.min()) / (X_test.max() - X_test.min())

# Train
nn.train(X_train, y_train_one_hot, epochs=300)

#Accuracy on the test set
test_accuracy = nn.accuracy(X_test, y_test)
print(f'Test Accuracy: {test_accuracy * 100:.2f}%')
```

Σχήμα 9: Predict and accuracy

```python
# Τα διάφορα διαγράμματα

# Διάγραμμα accuracy
plt.figure(figsize=(10, 6))
plt.plot(nn.train_accuracies, label='Training Accuracy')
plt.plot(nn.test_accuracies, label='Test Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Test Accuracy over Epochs')
plt.legend()

plt.tight_layout()
plt.show()


# Διάγραμμα απώλειας
plt.figure(figsize=(10, 6))
plt.plot(nn.train_losses, label='Training Loss')
plt.plot(nn.test_losses, label='Test Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Test Loss over Epochs')
plt.legend()

plt.tight_layout()
plt.show()
```

Σχήμα 10: Διαγράμματα

```python
# Διάγραμμα σύγχυσης
y_test_pred = nn.predict(X_test)
cm = confusion_matrix(y_test, y_test_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=trainset.classes)
disp.plot(cmap='Blues', xticks_rotation='vertical')
plt.title("Confusion Matrix")
plt.show()

# Σωστά παραδείγματα
correct_indices = np.where(y_test == y_test_pred)[0]
incorrect_indices = np.where(y_test != y_test_pred)[0]

fig1, axs1 = plt.subplots(2, 2, figsize=(10, 6))  # Two rows and two columns for correct examples
fig1.suptitle("Correct Classifications")

for i in range(4):  # 4 σωστά παραδείγματα
    idx_correct = correct_indices[np.random.randint(0, len(correct_indices))]
    img = X_test[idx_correct].reshape(3, 32, 32).transpose(1, 2, 0) * 0.5 + 0.5
    row = i // 2
    col = i % 2
    axs1[row, col].imshow(img)
    axs1[row, col].set_title(f"Correct: {trainset.classes[y_test[idx_correct]]}")
    axs1[row, col].axis('off')
```

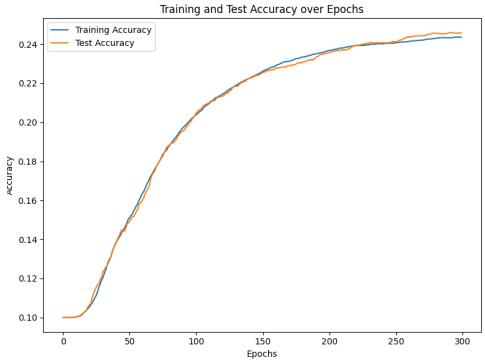
Σχήμα 11: Διαγράμματα

```python
# Τυχαία παραδείγματα - Incorrect Classifications
fig2, axs2 = plt.subplots(2, 2, figsize=(10, 6))
fig2.suptitle("Incorrect Classifications")

for i in range(4):  # 4 λάθος παραδείγματα
    idx_incorrect = incorrect_indices[np.random.randint(0, len(incorrect_indices))]
    img = X_test[idx_incorrect].reshape(3, 32, 32).transpose(1, 2, 0) * 0.5 + 0.5
    row = i // 2
    col = i % 2
    axs2[row, col].imshow(img)
    axs2[row, col].set_title(f"True: {trainset.classes[y_test[idx_incorrect]]}\nPred: {trainset.classes[y_test_pred[idx_incorrect]]}")
    axs2[row, col].axis('off')

# Show both plots
plt.tight_layout()
plt.show()
```

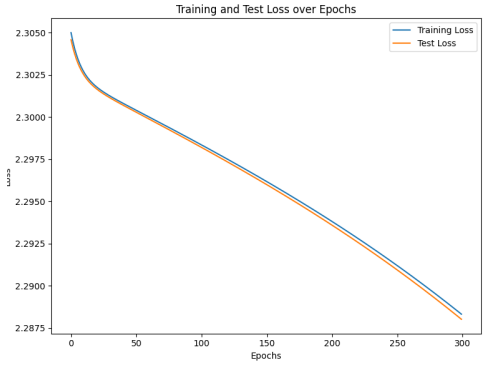
Σχήμα 12: Διαγράμματα

Ο αλγόριθμος έτρεξε για διάφορες σταθερές τιμές του learning rate καθώς και για μεταβλητό learning rate. Σε αυτή την ενότητα θα παρουσιαστούν τα αποτελέσματα για τους διαφορετικούς συνδιασμούς learning rate, τις διαφορετικές συναρτήσεις ενεργοποίησης και διαφορετικούς αριθμούς εσωτερικών νευρώνων. Ο αλγόριθμος έτρεξε για 300 εποχές.

- Η περίπτωση που χρησιμοποιήθηκε σιγμοειδής και $learning_rate = 0.01$ και 256 νευρώνων στο κρυφό επίπεδο. Το τελικό ποσοστό επιτυχίας ειναι 24,58% και ο χρόνος εκτέλεσης 2472,48 δευτερόλεπτα. Παρακάτω φαίνονται τα διαγράμματα που φαίνονται πως εξελίχθηκε το ποσοστό επιτυχίας και η απώλεια κατα τη διάρκεια των εποχών.

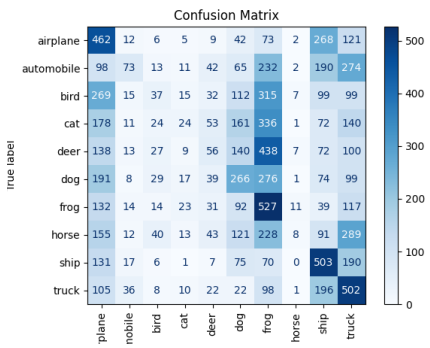


Σχήμα 13: Το ποσοστό επιτυχίας

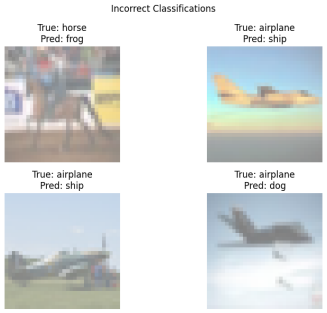


Σχήμα 14: Η απώλεια

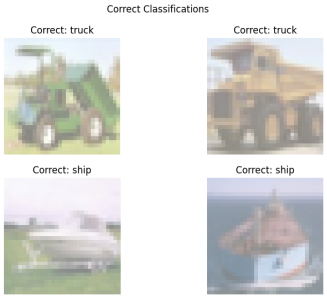
Στα επόμενα σχήματα φαίνεται το confusion diagram καθώς και παραδείγματα σωστής και εσφαλμένης χρήσης του αλγορίθμου.



Σχήμα 15: confusion diagram

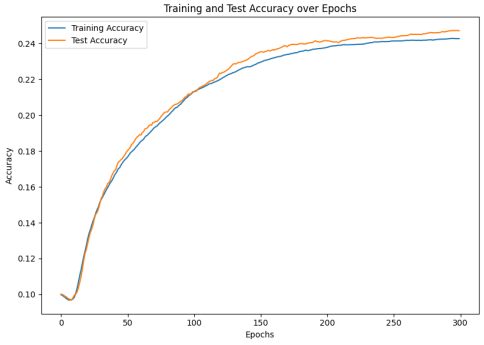


Σχήμα 16: Παραδείγματα εσφαλμένης κατηγοριοποίησης

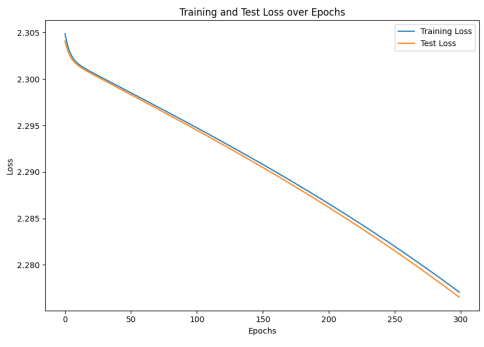


Σχήμα 17: Παραδείγματα σωστής κατηγοριοποίησης

- Η περίπτωση που χρησιμοποιήθηκε σιγμοειδής και $learninRate = 0.01$ και 512 νευρώνων στο κρυφό επίπεδο. Το τελικό ποσοστό επιτυχίας ειναι 24,71% και ο χρόνος εκτέλεσης 3186,80 δευτερόλεπτα. Παρακάτω φαίνονται τα διαγράμματα που φαίνονται πως εξελίχθηκε το ποσοστό επιτυχίας και η απώλεια κατα τη διάρκεια των εποχών.

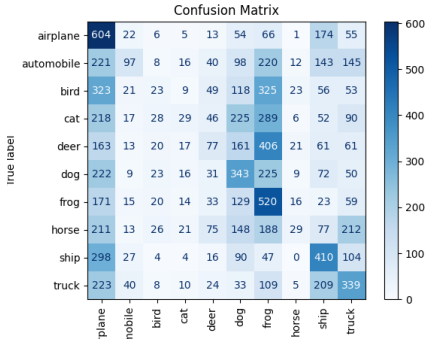


Σχήμα 18: Το ποσοστό επιτυχίας

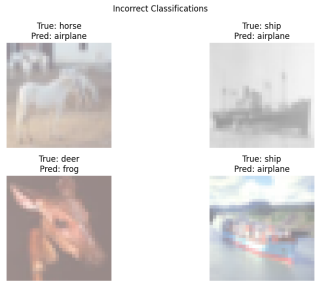


Σχήμα 19: Η απώλεια

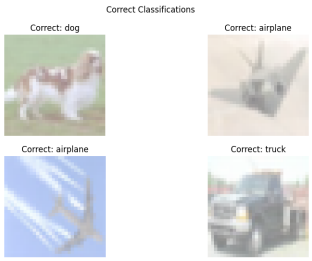
Στα επόμενα σχήματα φαίνεται το confusion diagram καθώς και παραδείγματα σωστής και εσφαλμένης χρήσης του αλγορίθμου.



Σχήμα 20: confusion diagram

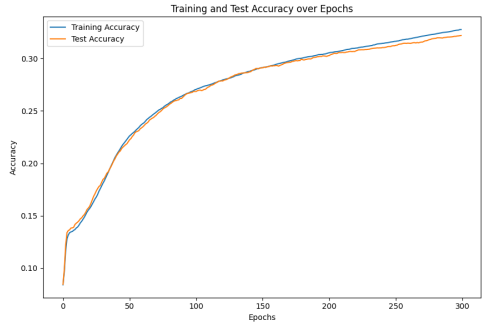


Σχήμα 21: Παραδείγματα εσφαλμένης κατηγοριοποίησης

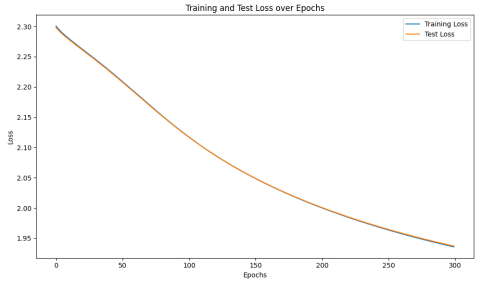


Σχήμα 22: Παραδείγματα σωστής κατηγοριοποίησης

- Η περίπτωση που χρησιμοποιήθηκε συνάρτηση relu και $learningRate = 0.02$ και 1024 νευρώνων στο κρυφό επίπεδο. Το τελικό ποσοστό επιτυχίας ειναι 32,2% και ο χρόνος εκτέλεσης 5144,59 δευτερόλεπτα. Παρακάτω φαίνονται τα διαγράμματα που φαίνονται πως εξελίχθηκε το ποσοστό επιτυχίας και η απώλεια κατα τη διάρκεια των εποχών.

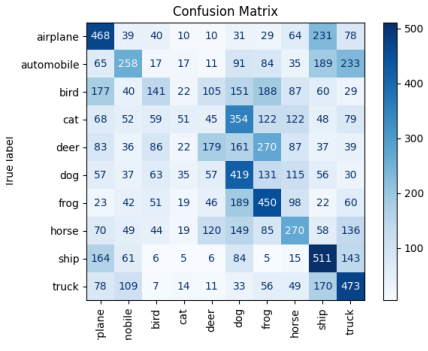


Σχήμα 23: Το ποσοστό επιτυχίας

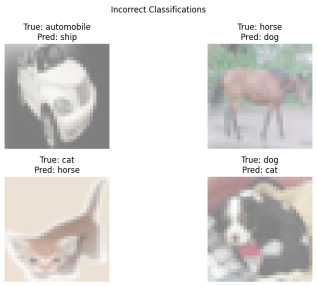


Σχήμα 24: Η απώλεια

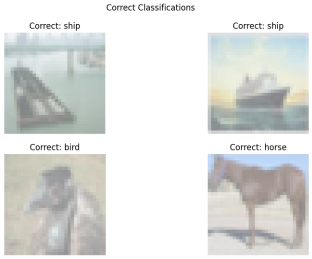
Στα επόμενα σχήματα φαίνεται το confusion diagram καθώς και παραδείγματα σωστής και εσφαλμένης χρήσης του αλγορίθμου.



Σχήμα 25: confusion diagram

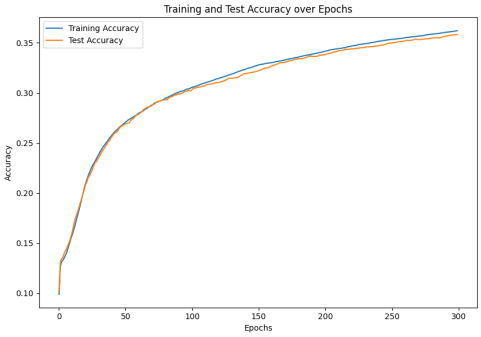


Σχήμα 26: Παραδείγματα εσφαλμένης κατηγοριοποίησης

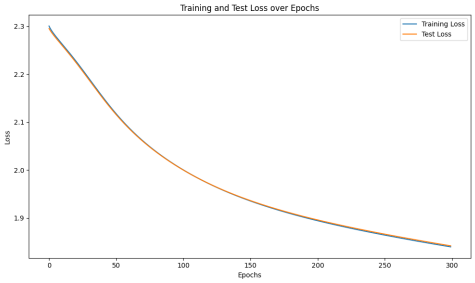


Σχήμα 27: Παραδείγματα σωστής κατηγοριοποίησης

- Η περίπτωση που χρησιμοποιήθηκε συνάρτηση relu και $learningRate = 0.04$ και 1024 νευρώνων στο κρυφό επίπεδο. Το τελικό ποσοστό επιτυχίας ειναι 35,82% και ο χρόνος εκτέλεσης 6099,04 δευτερόλεπτα. Παρακάτω φαίνονται τα διαγράμματα που φαίνονται πως εξελίχθηκε το ποσοστό επιτυχίας και η απώλεια κατα τη διάρκεια των εποχών.

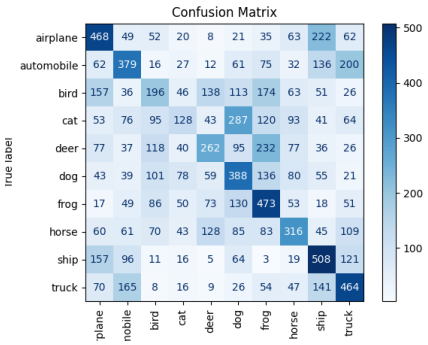


Σχήμα 28: Το ποσοστό επιτυχίας

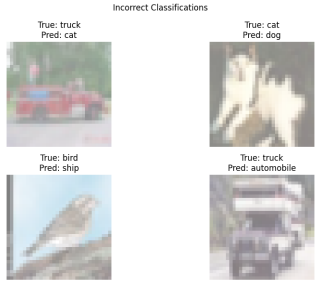


Σχήμα 29: Η απώλεια

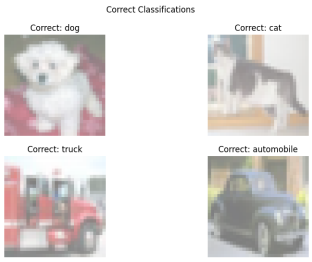
Στα επόμενα σχήματα φαίνεται το confusion diagram καθώς και παραδείγματα σωστής και εσφαλμένης χρήσης του αλγορίθμου.



Σχήμα 30: confusion diagram

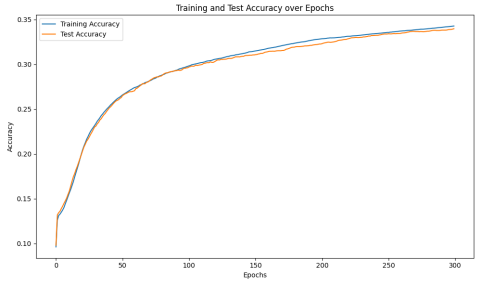


Σχήμα 31: Παραδείγματα εσφαλμένης κατηγοριοποίησης

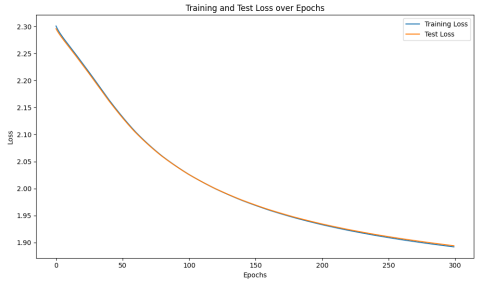


Σχήμα 32: Παραδείγματα σωστής κατηγοριοποίησης

- Η περίπτωση που χρησιμοποιήθηκε συνάρτηση relu και το $learning\_rate = 0.04$ μειόνεται εκθετικά κατα έναν σταθερό παράγοντα learning_rate = learning_rate × decay_factor.Η λογική πίσω απο αυτό είναι ότι στην αρχή απαιτείται ταχύτερη μάθηση ένω στη συνέχεια πιο λεπτομερής. Στο κρυφό επίπεδο υπάρχουν 1024 νευρώνες. Το τελικό ποσοστό επιτυχίας ειναι 33,97% και ο χρόνος εκτέλεσης 4711,92 δευτερόλεπτα. Παρακάτω φαίνονται τα διαγράμματα που φαίνονται πως εξελίχθηκε το ποσοστό επιτυχίας και η απώλεια κατα τη διάρκεια των εποχών.

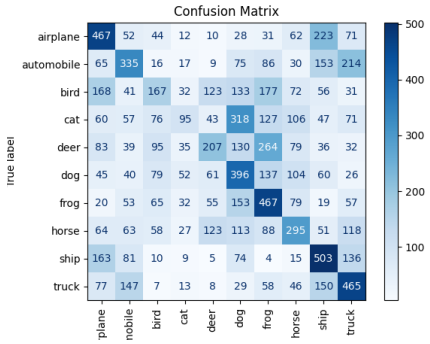


Σχήμα 33: Το ποσοστό επιτυχίας

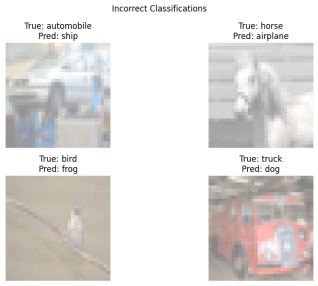


Σχήμα 34: Η απώλεια

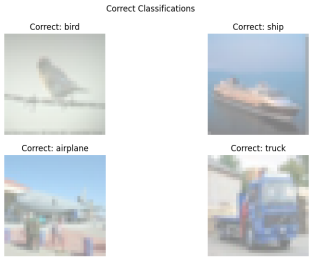
Στα επόμενα σχήματα φαίνεται το confusion diagram καθώς και παραδείγματα σωστής και εσφαλμένης χρήσης του αλγορίθμου.



Σχήμα 35: confusion diagram



Σχήμα 36: Παραδείγματα εσφαλμένης κατηγοριοποίησης



Σχήμα 37: Παραδείγματα σωστής κατηγοριοποίησης

- Παρατηρούμε γενίκα ότι τα καλύτερα αποτελέσματα λαμβάνονται για μεγαλύτερο learning rate και περισσότερους αριθμούς νευρώνων. Βλέπουμε ότι τα ποσοστά επιτυχίας είναι παρόμοια για το training και το test set και αυξάνονται σταδιακά με το χρόνο.

V . Νευρωνικό με συνελικτικά επίπεδα

Τα προβλήματα κατηγοριοποίησης επιλύονται αποτελεσματικότερα χρησιμοποιώντας νευρωνικά δίκτυα με συνελικτικά επίπεδα. Για τον λόγω αυτό υλοποιήθηκε ένα νευρωνικό δίκτυο χρησιμοποιώντας συναρτήσεις της βιβλιοθήκης pytorch.

Η αρχιτεκτονική που χρησιμοποιείται στο μοντέλο αποτελείται από ένα συνελικτικό νευρωνικό δίκτυο (**Convolutional Neural Network - CNN**) με τρία συνελικτικά επίπεδα (convolutional layers), τα οποία συνοδεύονται από επίπεδα υποδειγματοληψίας (pooling layers). Αναλυτικά:

- Το πρώτο συνελικτικό επίπεδο (Conv2D) δέχεται εικόνες εισόδου μεγέθους $32 \times 32 \times 3$ (RGB) και εφαρμόζει 32 φίλτρα με πυρήνα μεγέθους 3×3, διατηρώντας το μέγεθος της εξόδου ίδιο μέσω παδδινγ.

- Το δεύτερο συνελικτικό επίπεδο εφαρμόζει 64 φίλτρα με πυρήνα 3×3, ακολουθούμενο από ένα επίπεδο max pooling2×2 για μείωση της χωρικής ανάλυσης.
- Το τρίτο συνελικτικό επίπεδο εφαρμόζει 128 φίλτρα, επίσης με πυρήνα 3×3.

Μετά τη συνελικτική ενότητα, το δίκτυο περιλαμβάνει ένα πλήρως συνδεδεμένο επίπεδο (fully connected layer) με 512 νευρώνες και λειτουργία ενεργοποίησης ReLU, ακολουθούμενο από ένα επίπεδο **Dropout** με ποσοστό 50% για την αποφυγή υπερεκμάθησης. Στο τελικό επίπεδο εξόδου (output layer), υπάρχει ένας πλήρως συνδεδεμένος πυρήνας 10 νευρώνων (ένας για κάθε κατηγορία της CIFAR-10), με χρήση της συνάρτησης softmax για ταξινόμηση.

Το δίκτυο εκπαιδεύεται με την Cross Entropy Loss ως συνάρτηση κόστους και χρησιμοποιεί τον Adam Optimizer με αρχικό ρυθμό εκμάθησης 0.0001. Ο Adam Optimizer χρησιμοποιεί την πρώτη (μέσος όρος gradient) και τη δεύτερη ροπή (τετραγωνικός μέσος gradient) των γραδιεντς για την προσαρμογή του ρυθμού εκμάθησης. Η εκπαίδευση επωφελείται από τη χρήση δεδομένων με κανονικοποίηση και τεχνικές αύξησης δεδομένων (*data augmentation*) όπως τυχαία περιστροφή (*random horizontal flip*) και αποκοπή (*random crop*).

Η υλοποίηση φαίνεται στις παρακάτω εικόνες

```python
#Οι βιβλιοθήκες που χρησιμοποιήθηκαν
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import time
import random

transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),  #Random flip
    transforms.RandomCrop(32, padding=4),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  # Normalization
])

batch_size = 64

# Load the training set
trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size, shuffle=True)

# Load the test set
testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size, shuffle=False)
```

Σχήμα 38: Φόρτωση βιβλιοθηκών και data set

```python
# Το CNN
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        #Συνελικτικά επίπεδα
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0) #Max pooling
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
        #Συνδεδεμένα Επίπεδα
        self.fc1 = nn.Linear(128 * 4 * 4, 512)
        self.fc2 = nn.Linear(512, 10)
        self.dropout = nn.Dropout(0.5) #Drop out για την αποφυγή του overfitting
        self.relu = nn.ReLU() # Συνάρτηση ενεργοποίησης στην έξοδο

        #Εμπρόσθιο πέρασμα
    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
        x = self.pool(self.relu(self.conv3(x)))
        x = x.view(-1, 128 * 4 * 4)  # Flatten for the fully connected layer
        x = self.dropout(self.relu(self.fc1(x)))
        x = self.fc2(x)
        return x
```

Σχήμα 39: Ορισμός νευρωνικού

```python
# Σε περίπτωση που τρέξει σε gpu
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = SimpleCNN().to(device)
criterion = nn.CrossEntropyLoss() #cross entropy loss
optimizer = optim.Adam(model.parameters(), lr=0.0001) #Adam optimizer

train_losses = []
test_losses = []
train_accuracies = []
test_accuracies = []

#Εκπαίδευση του μοντέλου
num_epochs = 10
start_time = time.time()#Ξεκίνημα χρονομέτρησης
for epoch in range(num_epochs):
    epoch_start_time = time.time()  # Ξεκίνημα του χρονομέτρου κάθε εποχής
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for i, (inputs, labels) in enumerate(trainloader, 0):
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()  # Μηδενίζω τις παραμέτρους των παραγώγων
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()  # Backpropagation
        optimizer.step()  # Ανανέωση βαρών
```

Σχήμα 40: Αρχή της εκπαίδευσης

```python
        running_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    train_losses.append(running_loss / len(trainloader))
    train_accuracies.append(100 * correct / total)

    # Evaluate on test set
    model.eval()
    test_loss = 0.0
    test_correct = 0
    test_total = 0
    with torch.no_grad():
        for inputs, labels in testloader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            test_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            test_total += labels.size(0)
            test_correct += (predicted == labels).sum().item()

    test_losses.append(test_loss / len(testloader))
    test_accuracies.append(100 * test_correct / test_total)

    epoch_end_time = time.time()  # Τέλος χρονομέτρησης εποχής
    epoch_time = epoch_end_time - epoch_start_time #Χρόνος εποχής
    print(
        f'Epoch {epoch + 1}/{num_epochs}, Train Loss: {train_losses[-1]:.4f}, Test Loss: {test_losses[-1]:.4f}, Train Acc: {train_accuracies[-1]:.2f}%, Test Acc: {test_accuracies[-
```

Σχήμα 41: Συνέχεια της εκπαίδευσης

```python
end_time = time.time()#Τέλος συνολικής χρονομέτρησης
time = end_time - start_time #Συνολικός Χρόνος
print(f'Total Training Time: {time:.2f} sec')
# Διαγράμματα όπως και στην περίπτωση του συνελικτικού
plt.figure(figsize=(8, 6))
plt.plot(train_accuracies, label='Train Accuracy')
plt.plot(test_accuracies, label='Test Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy (%)')
plt.title('Accuracy over Epochs')
plt.legend()
plt.show()

plt.figure(figsize=(8, 6))
plt.plot(train_losses, label='Train Loss')
plt.plot(test_losses, label='Test Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss over Epochs')
plt.legend()
plt.show()

y_test = []
y_test_pred = []
```

Σχήμα 42: Διαγράμματα

```python
model.eval()
with torch.no_grad():
    for inputs, labels in testloader:
        inputs = inputs.to(device)
        outputs = model(inputs)
        _, predicted = torch.max(outputs, 1)
        y_test.extend(labels.cpu().numpy())
        y_test_pred.extend(predicted.cpu().numpy())

cm = confusion_matrix(y_test, y_test_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=testset.classes)
disp.plot(cmap='Blues', xticks_rotation='vertical')
plt.title("Confusion Matrix")
plt.show()


def denormalize(image):
    image = image * 0.5 + 0.5
    return np.transpose(image.cpu().numpy(), (1, 2, 0))


correct_indices = [i for i, (true, pred) in enumerate(zip(y_test, y_test_pred)) if true == pred]
incorrect_indices = [i for i, (true, pred) in enumerate(zip(y_test, y_test_pred)) if true != pred]
```

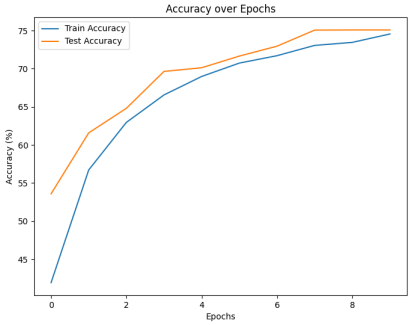
Σχήμα 43: Διαγράμματα

```python
fig1, axs1 = plt.subplots(2, 2, figsize=(10, 6))
fig1.suptitle("Correct Classifications")
for i in range(4):
    idx = random.choice(correct_indices)
    img, label = testset[idx]
    axs1[i // 2, i % 2].imshow(denormalize(img))
    axs1[i // 2, i % 2].set_title(f"Class: {testset.classes[label]}")
    axs1[i // 2, i % 2].axis('off')

fig2, axs2 = plt.subplots(2, 2, figsize=(10, 6))
fig2.suptitle("Incorrect Classifications")
for i in range(4):
    idx = random.choice(incorrect_indices)
    img, label = testset[idx]
    pred_label = y_test_pred[idx]
    axs2[i // 2, i % 2].imshow(denormalize(img))
    axs2[i // 2, i % 2].set_title(f"True: {testset.classes[label]}, Pred: {testset.classes[pred_label]}")
    axs2[i // 2, i % 2].axis('off')

plt.tight_layout()
plt.show()
```

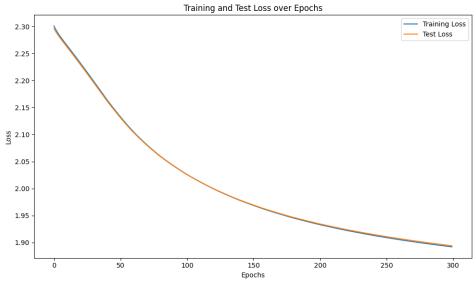
Σχήμα 44: Διαγράμματα

Παρακάτω θα παρουσιάσουμε τα αποτελέσματα του νευρωνικού δικτύου για διαφορετικό learning rate και batch size για 10 εποχές.

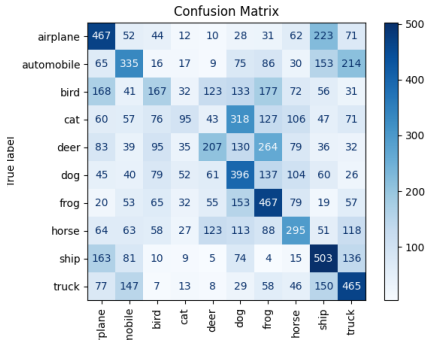
- Για batch_size=32 και learning_rate=0.001 το ποσοστό επιτυχίας της κατηγοριοποίησης είναι 75,43% και ο χρόνος εκτέλεσης του αλγορίθμου είναι 1231,97 δευτερόλεπτα. Τα διαγράμματα accuracy, loss και confusion φαίνονται στα παρακάτω σχήματα.



Σχήμα 45: Το ποσοστό επιτυχίας

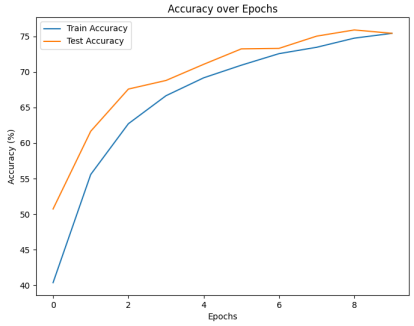


Σχήμα 46: Η απώλεια

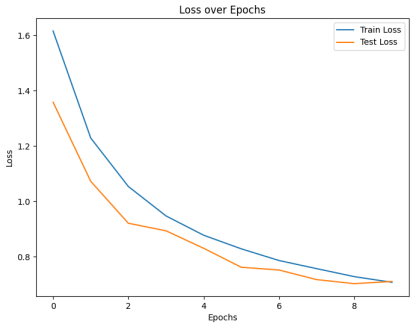


Σχήμα 47: confusion diagram

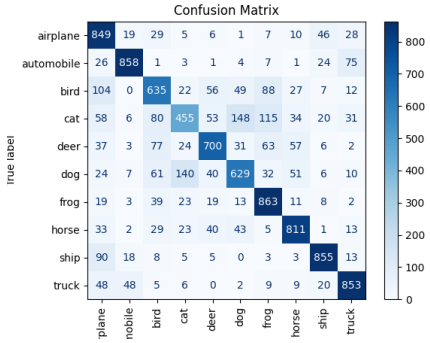
- Για batch_size=64 και learning_rate=0.001 το ποσοστό επιτυχίας της κατηγοριοποίησης είναι 75,43% και ο χρόνος εκτέλεσης του αλγορίθμου είναι 1231,97 δευτερόλεπτα. Τα διαγράμματα accuracy, loss και confusion φαίνονται στα παρακάτω σχήματα.



Σχήμα 48: Το ποσοστό επιτυχίας

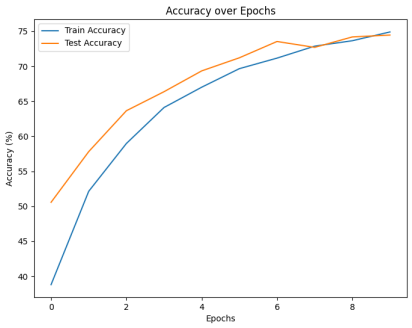


Σχήμα 49: Η απώλεια

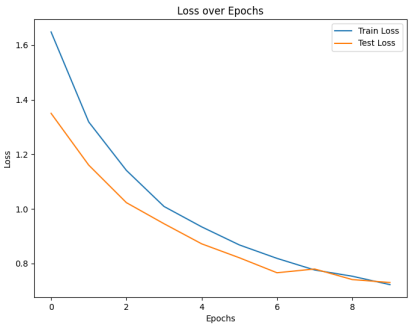


Σχήμα 50: confusion diagram

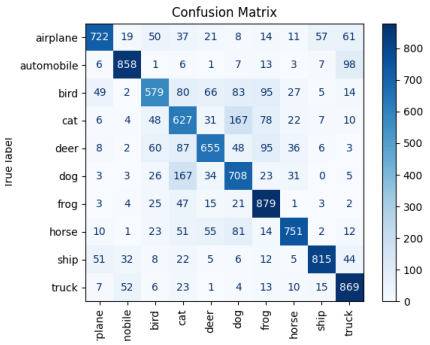
- Για batch_size=128 και learning_rate=0.001 το ποσοστό επιτυχίας της κατηγοριοποίησης είναι 74,46% και ο χρόνος εκτέλεσης του αλγορίθμου είναι 699,31 δευτερόλεπτα. Τα διαγράμματα accuracy, loss και confusion φαίνονται στα παρακάτω σχήματα.



Σχήμα 51: Το ποσοστό επιτυχίας

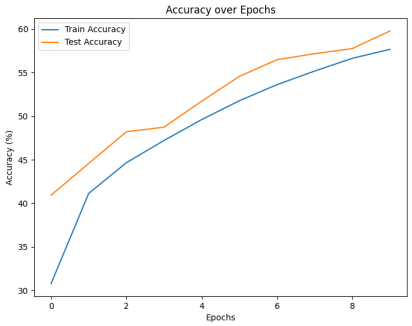


Σχήμα 52: Η απώλεια

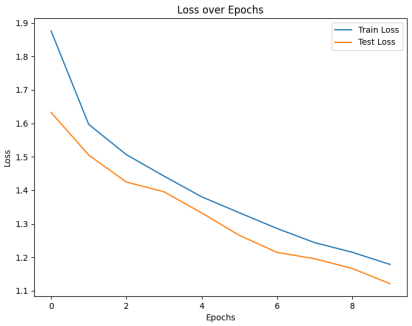


Σχήμα 53: confusion diagram

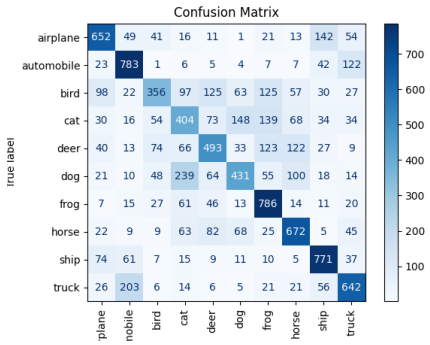
- Για batch_size=64 και learning_rate=0.0001 το ποσοστό επιτυχίας της κατηγοριοποίησης είναι 51,02% και ο χρόνος εκτέλεσης του αλγορίθμου είναι 781,05 δευτερόλεπτα. Τα διαγράμματα accuracy, loss και confusion φαίνονται στα παρακάτω σχήματα.



Σχήμα 54: Το ποσοστό επιτυχίας

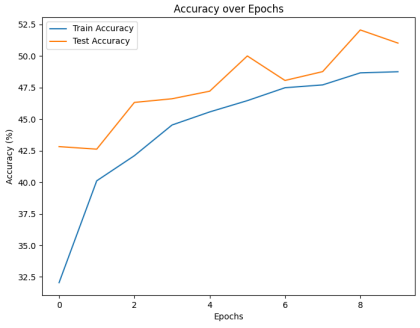


Σχήμα 55: Η απώλεια

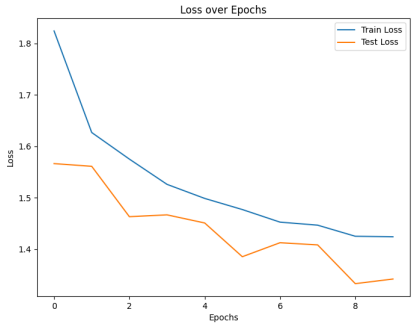


Σχήμα 56: confusion diagram

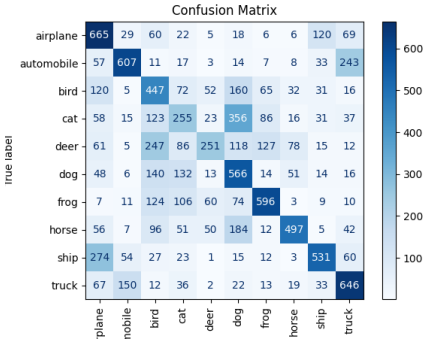
- Για batch_size=64 και learning_rate=0.005 το ποσοστό επιτυχίας της κατηγοριοποίησης είναι 51,02% και ο χρόνος εκτέλεσης του αλγορίθμου είναι 671,44 δευτερόλεπτα. Τα διαγράμματα accuracy, loss και confusion φαίνονται στα παρακάτω σχήματα.



Σχήμα 57: Το ποσοστό επιτυχίας



Σχήμα 58: Η απώλεια



Σχήμα 59: confusion diagram

- Παρατηρούμε ότι στους συνδιασμούς που δοκιμάσαμε η αλλαγή στο batch size δεν επιφέρει σημαντικές διαφορές στην ακρίβεια του αλγορίθμου. Αντίθετα, αύξηση και μείωση του learning rate ρίχει την απόδοση.

VI . Συμπεράσματα

Το πλήρες συνδεδεμένο νευρωνικό δίκτυο είχε παρόμοια ποσοστά επιτυχίας με τους κατηγοριοποιητές πλησιέστερου γείτονα και πλησιέστερου κέντρου. Αντίθετα, το νευρωνικό με τα συνελικτικά επίπεδα παρουσίασε σημαντικά καλυτερά ποσοστά επιτυχούς κατηγοριοποίησης. Αυτο οφείλεται τόσο στο ότι η υλοποιήση με βιβλιοθήκες είναι η βέλτιστη όσο και στο ότι το πρόβλημα κατηγοριοποίησης εικόνων επιλύεται καλύτερα με συνελικτικό νευρωνικό επειδη λαμβάνει καλυτερα υπ΄ όψη τυχούσες διαφοροποιήσης στην θέση του υπο αναγνώριση αντικειμένου