ΜΥΕ046 – Υπολογιστική Όραση: Χειμερινό εξάμηνο 2024-2025

Εργασία: 30% του συνολικού βαθμού

Διδάσκων: Άγγελος Γιώτης

- ΠΑΡΑΔΟΣΗ: **Τρίτη, 14 Ιανουαρίου, 2025 23:59**

Γενικές Οδηγίες

Απαντήστε στα παρακάτω ζητήματα χρησιμοποιώντας Python στο συνημμένο σημειωματάριο Jupyter και ακολουθήστε τις παρακάτω οδηγίες:

- Οι ασκήσεις είναι **ατομικές** - δεν επιτρέπεται η μεταξύ σας συνεργασία για την υλοποίηση/παράδοσή τους.
- **Δεν** επιτρέπεται να χρησιμοποιήσετε κώδικα που τυχόν θα βρείτε στο διαδίκτυο (είτε αυτούσιο, είτε **παραγόμενο από AI**). Η χρήση κώδικα τρίτων θα έχει σαν αποτέλεσμα τον αυτόματο μηδενισμό σας.
- Όλες οι λύσεις πρέπει να είναι γραμμένες σε αυτό το σημειωματάριο `Jupyter notebook` .
- **Εάν** ένα ζήτημα περιλαμβάνει θεωρητική ερώτηση, η απάντηση θα **πρέπει** να συμπεριληφθεί στο τέλος του ζητήματος, σε ξεχωριστό "Markdown" κελί.
- Ο κώδικάς σας πρέπει να σχολιαστεί εκτενώς! Καλά σχολιασμένος κώδικας θα συνεκτιμηθεί στην αξιολόγησή σας.
- Αφού ολοκληρώσετε (υλοποιήσετε και εκτελέσετε) τις απαντήσεις σας στο σημειωματάριο (notebook), εξαγάγετε το notebook ως **PDF** και υποβάλετε, τόσο το σημειωματάριο όσο και το PDF (δηλαδή τα αρχεία `.ipynb` και `.pdf`) στο `turnin` του μαθήματος, μαζί με ένα συνοδευτικό αρχείο `onoma.txt` που θα περιέχει το ον/μο σας και τον Α.Μ. σας. Μια καλή πρακτική για την αποφυγή προβλημάτων απεικόνισης, π.χ., περικοπής εικόνων/κώδικα στα όρια της σελίδας, είναι η μετατροπή του `.ipynb` σε HTML και μετά η

αποθήκευση του HTML αρχείου ως PDF.

- Οι απαντήσεις θα παραδοθούν με την εντολή: **turnin assignment@mye046 onoma.txt assignment.ipynb assignment.pdf**
- Μπορείτε να χρησιμοποιήσετε βασικά πακέτα γραμμικής άλγεβρας (π.χ. `NumPy` , `SciPy`), αλλά δεν επιτρέπεται να χρησιμοποιείτε τα πακέτα/βιβλιοθήκες που επιλύουν άμεσα τα προβλήματα, εκτός και αν αναφέρεται διαφορετικά η χρήση συγκεκριμένου πακέτου σε κάποιο ζήτημα. Αν δεν είστε βέβαιοι για κάποιο συγκεκριμένο πακέτο/βιβλιοθήκη ή συνάρτηση που θα χρησιμοποιήσετε, μη διστάσετε να ρωτήσετε τον διδάσκοντα.
- Συνιστάται ιδιαίτερα να αρχίσετε να εργάζεστε στις ασκήσεις σας το συντομότερο δυνατό!

Late Policy: Εργασίες που υποβάλλονται καθυστερημένα θα λαμβάνουν μείωση βαθμού 10% για κάθε 24 ώρες καθυστέρησης. Οι εργασίες δεν θα γίνονται δεκτές 96 ώρες (4 ημέρες) μετά την προθεσμία παράδοσης. Για παράδειγμα, παράδοση της εργασίας 2 ημέρες μετά την προθεσμία βαθμολογείται με άριστα το 24 (από 30).

Intro to Google Colab, Jupyter Notebook - JupyterLab, Python

Εισαγωγή

- Η Εργασία του μαθήματος MYE046-Υπολογιστική Όραση περιλαμβάνει 2 Ασκήσεις στο αρχείο `assignment.ipynb` , το οποίο απαιτεί περιβάλλον Jupyter Notebook ή JupyterLab για προβολή και επεξεργασία, **είτε τοπικά** (local machine) στον υπολογιστή σας, **είτε μέσω της υπηρεσίας** νέφους Google Colab ή Colaboratory.

Working remotely on Google Colaboratory

Το Google Colaboratory είναι ένας συνδυασμός σημειωματαρίου Jupyter και Google Drive. Εκτελείται εξ' ολοκλήρου στο cloud και έρχεται προεγκατεστημένο με πολλά πακέτα (π.χ. PyTorch και Tensorflow), ώστε όλοι να έχουν πρόσβαση στις ίδιες εξαρτήσεις/βιβλιοθήκες. Ακόμη πιο ενδιαφέρον είναι το γεγονός ότι το Colab επωφελείται από την ελεύθερη πρόσβαση σε επιταχυντές υλικού (π.χ. κάρτες γραφικών) όπως οι GPU (K80, P100) και οι TPU.

- Requirements:

Για να χρησιμοποιήσετε το Colab, πρέπει να έχετε λογαριασμό Google με συσχετισμένο Google Drive. Υποθέτοντας ότι έχετε και τα δύο (ο

ακαδημαϊκός σας λογαριασμός είναι λογαριασμός google), μπορείτε να συνδέσετε το Colab στο Drive σας με τα ακόλουθα βήματα:

1. Κάντε κλικ στον τροχό στην επάνω δεξιά γωνία (στο Google Drive) και επιλέξτε `Ρυθμίσεις`.
2. Κάντε κλικ στην καρτέλα `Διαχείριση εφαρμογών`.
3. Στο επάνω μέρος, επιλέξτε `Σύνδεση περισσότερων εφαρμογών` που θα εμφανίσουν ένα παράθυρο του `GSuite Marketplace`.
4. Αναζητήστε το `Colab` και, στη συνέχεια, κάντε κλικ στην `Προσθήκη` (install).

- Workflow:

Η εργασία στη σελίδα ecourse του μαθήματος παρέχει έναν σύνδεσμο λήψης σε ένα αρχείο `assignment.zip` που περιέχει:

1. `images/`, φάκελος με ενδεικτικές εικόνες των παρακάτω ζητημάτων.
2. `assignment.ipynb`, το σημειωματάριο jupyter στο οποίο θα εργαστείτε και θα παραδώσετε.
3. `tutorial1_pytorch_introduction.ipynb`, που περιλαμβάνει στοιχειώδη παραδείγματα με χρήση της βιλβιοθήκης βαθιάς μάθησης `PyTorch` (αφορά στη 2η εργασία).
4. `Σημειώσεις PCA-SVD.pdf`, σημειώσεις που σχετίζονται με το ζήτημα **1.6** της **1ης** άσκησης.
5. `Σημειώσεις CNN.pdf`, σημειώσεις που σχετίζονται με το ζήτημα **2.5** της **2ης** άσκησης.

- Βέλτιστες πρακτικές:

Υπάρχουν μερικά πράγματα που πρέπει να γνωρίζετε όταν εργάζεστε με την υπηρεσία Colab. Το πρώτο πράγμα που πρέπει να σημειωθεί είναι ότι οι πόροι δεν είναι εγγυημένοι (αυτό είναι το τίμημα της δωρεάν χρήσης). Εάν είστε σε αδράνεια για ένα συγκεκριμένο χρονικό διάστημα ή ο συνολικός χρόνος σύνδεσής σας υπερβαίνει τον μέγιστο επιτρεπόμενο χρόνο (~12 ώρες), το Colab VM θα αποσυνδεθεί. Αυτό σημαίνει ότι οποιαδήποτε μη αποθηκευμένη πρόοδος θα χαθεί. Έτσι, φροντίστε να αποθηκεύετε συχνά την υλοποίησή σας ενώ εργάζεστε.

- Χρήση GPU:

Η χρήση μιας GPU απαιτεί πολύ απλά την αλλαγή του τύπου εκτέλεσης (runtime) στο Colab. Συγκεκριμένα, κάντε κλικ `Runtime -> Change runtime type -> Hardware Accelerator -> GPU` και το στιγμιότυπο εκτέλεσής σας Colab θα υποστηρίζεται αυτόματα από επιταχυντή υπολογισμών GPU (αλλαγή τύπου χρόνου εκτέλεσης σε GPU ή TPU). Στην παρούσα εργασία, **δεν** θα χρειαστεί η χρήση GPU.

Working locally on your machine

Linux

Εάν θέλετε να εργαστείτε τοπικά στον Η/Υ σας, θα πρέπει να χρησιμοποιήσετε ένα εικονικό περιβάλλον. Μπορείτε να εγκαταστήσετε ένα μέσω του Anaconda (συνιστάται) ή μέσω της native μονάδας `venv` της Python. Βεβαιωθείτε ότι χρησιμοποιείτε (τουλάχιστον) έκδοση Python 3.7.

- Εικονικό περιβάλλον Anaconda: Συνιστάται η χρήση της δωρεάν διανομής Anaconda, η οποία παρέχει έναν εύκολο τρόπο για να χειριστείτε τις εξαρτήσεις πακέτων. Μόλις εγκαταστήσετε το Anaconda, είναι εύχρηστο να δημιουργήσετε ένα εικονικό περιβάλλον για το μάθημα. Για να ρυθμίσετε ένα εικονικό περιβάλλον που ονομάζεται π.χ. `mye046` , εκτελέστε τα εξής στο τερματικό σας: `conda create -n mye046 python=3.7` (Αυτή η εντολή θα δημιουργήσει το περιβάλλον mye046 στη διαδρομή 'path/to/anaconda3/envs/') Για να ενεργοποιήσετε και να εισέλθετε στο περιβάλλον, εκτελέστε το `conda activate mye046` . Για να απενεργοποιήσετε το περιβάλλον, είτε εκτελέστε `conda deactivate mye046` είτε βγείτε από το τερματικό. Σημειώστε ότι κάθε φορά που θέλετε να εργαστείτε στην εργασία, θα πρέπει να εκτελείτε ξανά το `conda activate mye046` .

- Εικονικό περιβάλλον Python venv: Για να ρυθμίσετε ένα εικονικό περιβάλλον που ονομάζεται mye046, εκτελέστε τα εξής στο τερματικό σας: `python3.7 -m venv ~/mye046` Για να ενεργοποιήσετε και να εισέλθετε στο περιβάλλον, εκτελέστε το `source ~/mye046/bin/activate` . Για να απενεργοποιήσετε το περιβάλλον, εκτελέστε: `deactivate` ή έξοδο από το τερματικό. Σημειώστε ότι κάθε φορά που θέλετε να εργαστείτε για την άσκηση, θα πρέπει να εκτελείτε ξανά το `source ~/mye046/bin/activate` .

- Εκτέλεση Jupyter Notebook: Εάν θέλετε να εκτελέσετε το notebook τοπικά με το Jupyter, βεβαιωθείτε ότι το εικονικό σας περιβάλλον έχει εγκατασταθεί σωστά (σύμφωνα με τις οδηγίες εγκατάστασης που περιγράφονται παραπάνω για περιβάλλον linux), ενεργοποιήστε το και, στη συνέχεια, εκτελέστε `pip install notebook` για να εγκαταστήσετε το σημειωματάριο `Jupyter` . Στη συνέχεια, αφού κατεβάσετε και αποσυμπιέσετε το φάκελο της Άσκησης από τη σελίδα ecourse σε κάποιο κατάλογο της επιλογής σας, εκτελέστε `cd` σε αυτόν το φάκελο και στη συνέχεια εκτελέστε το σημειωματάριο `jupyter notebook` . Αυτό θα πρέπει να εκκινήσει αυτόματα έναν διακομιστή notebook στη διεύθυνση `http://localhost:8888` . Εάν όλα έγιναν σωστά, θα πρέπει να δείτε μια οθόνη που θα εμφανίζει όλα τα διαθέσιμα σημειωματάρια στον τρέχοντα κατάλογο, στην προκειμένη περίπτωση μόνο το `assignment.ipynb` (η εργασία σας). Κάντε κλικ στο `assignment.ipynb` και ακολουθήστε τις οδηγίες στο σημειωματάριο.

Windows

Τα πράγματα είναι πολύ πιο απλά στην περίπτωση που θέλετε να εργαστείτε τοπικά σε περιβάλλον `Windows` . Μπορείτε να εγκαταστήσετε την Anaconda για Windows και στη συνέχεια να εκτελέσετε το Anaconda Navigator αναζητώντας το απευθείας στο πεδίο

αναζήτησης δίπλα από το κουμπί `έναρξης` των Windows. Το εργαλείο αυτό παρέχει επίσης άμεσα προεγκατεστημένα, τα πακέτα λογισμικού Jupyter Notebook και JupyterLab τα οποία επιτρέπουν την προβολή και υλοποίηση του σημειοματαρίου Jupyter άμεσα και εύκολα (εκτελώντας το απευθείας από τη διαδρομή αρχείου που βρίσκεται). Ενδεχομένως, κατά την αποθήκευση/εξαγωγή του notebook `assignment.ipynb` σε `assignment.pdf`, να χρειαστεί η εγκατάσταση του πακέτου Pandoc universal document converter (εκτέλεση: `conda install -c conda-forge pandoc` μέσα από το command prompt του "activated" anaconda navigator). Εναλλακτικά, μπορεί να εκτυπωθεί ως PDF αρχείο (**βλ. Ενότητα**: Οδηγίες υποβολής).

Python

Θα χρησιμοποιήσουμε τη γλώσσα προγραμματισμού Python και στις 2 ασκήσεις, με μερικές δημοφιλείς βιβλιοθήκες (`NumPy`, `Matplotlib`) ενώ στη 2η άσκηση θα χρειαστεί και η βιβλιοθήκη βαθιάς μάθησης PyTorch. Αναμένεται ότι πολλοί από εσάς έχετε κάποια εμπειρία σε `Python` και `NumPy`. Και αν έχετε πρότερη εμπειρία σε `MATLAB`, μπορείτε να δείτε επίσης το σύνδεσμο NumPy for MATLAB users.

Άσκηση 1: Μηχανική Μάθηση [15 μονάδες]

Στην άσκηση αυτή θα υλοποιήσετε μια σειρά από παραδοσιακές τεχνικές μηχανικής μάθησης με εφαρμογή στην επίλυση προβλημάτων υπολογιστικής όρασης.

Ζήτημα 1.1: Αρχική Εγκατάσταση

Θα χρησιμοποιήσουμε την ενότητα Scikit-learn (Sklearn) για αυτή την άσκηση. Είναι μια από τις πιο χρήσιμες και ισχυρές βιβλιοθήκες για μηχανική μάθηση στην Python. Παρέχει μια επιλογή αποτελεσματικών εργαλείων για μηχανική μάθηση και στατιστική μοντελοποίηση, συμπεριλαμβανομένης της ταξινόμησης (classification), της παλινδρόμησης (regression), της ομαδοποίησης (clustering) και της μείωσης διάστασης (dimensionality reduction). Αυτό το πακέτο, το οποίο είναι σε μεγάλο βαθμό γραμμένο σε Python, βασίζεται στις βιβλιοθήκες NumPy, SciPy και Matplotlib.

Αρχικά καλούμε/εγκαθιστούμε τη βασική μονάδα της βιβλιοθήκης sklearn.

In [1]:
```
# Importing the scikit-learn library.
import sklearn
```

```
# Using the __version__ attribute to print
# the current version of the scikit-learn library installed.
sklearn.__version__
```

Out[1]: '1.6.0'

Ζήτημα 1.2: Λήψη συνόλου δεδομένων χειρόγραφων ψηφίων "MNIST" και απεικόνιση παραδειγμάτων [1 μονάδα]

Η βάση δεδομένων MNIST (Modified National Institute of Standards and Technology database) είναι ένα αρκετά διαδεδομένο σύνολο δεδομένων που αποτελείται από εικόνες χειρόγραφων ψηφίων, διαστάσεων 28x28 σε κλίμακα του γκρι. Για αυτό το ζήτημα, θα χρησιμοποιήσουμε το πακέτο Sklearn για να κάνουμε ταξινόμηση μηχανικής μάθησης στο σύνολο δεδομένων MNIST.

Το Sklearn παρέχει μια βάση δεδομένων MNIST χαμηλότερης ανάλυσης με εικόνες ψηφίων 8x8 pixel. Το πεδίο (attribute) `images` του συνόλου δεδομένων, αποθηκεύει πίνακες 8x8 τιμών κλίμακας του γκρι για κάθε εικόνα. Το πεδίο (attribute) `target` του συνόλου δεδομένων αποθηκεύει το ψηφίο που αντιπροσωπεύει κάθε εικόνα. Ολοκληρώστε τη συνάρτηση `plot_mnist_sample()` για να απεικονίσετε σε ένα σχήμα 2x5 ένα δείγμα εικόνας από κάθε μια κατηγορία (κάθε πλαίσιο του 2x5 σχήματος αντιστοιχεί σε ένα ψηφίο/εικόνα μιας κατηγορίας). Η παρακάτω εικόνα δίνει ένα παράδειγμα:

In [2]:
```
# Importing the numpy library, commonly used
# for numerical computations
import numpy as np

# Importing the matplotlib.pyplot module for
# reating visualizations and plots
import matplotlib.pyplot as plt

# Importing the datasets module from scikit-learn
# to access built-in datasets for machine learning tasks
from sklearn import datasets
```

In [3]:
```
# Download MNIST Dataset from Sklearn
digits = datasets.load_digits()
```

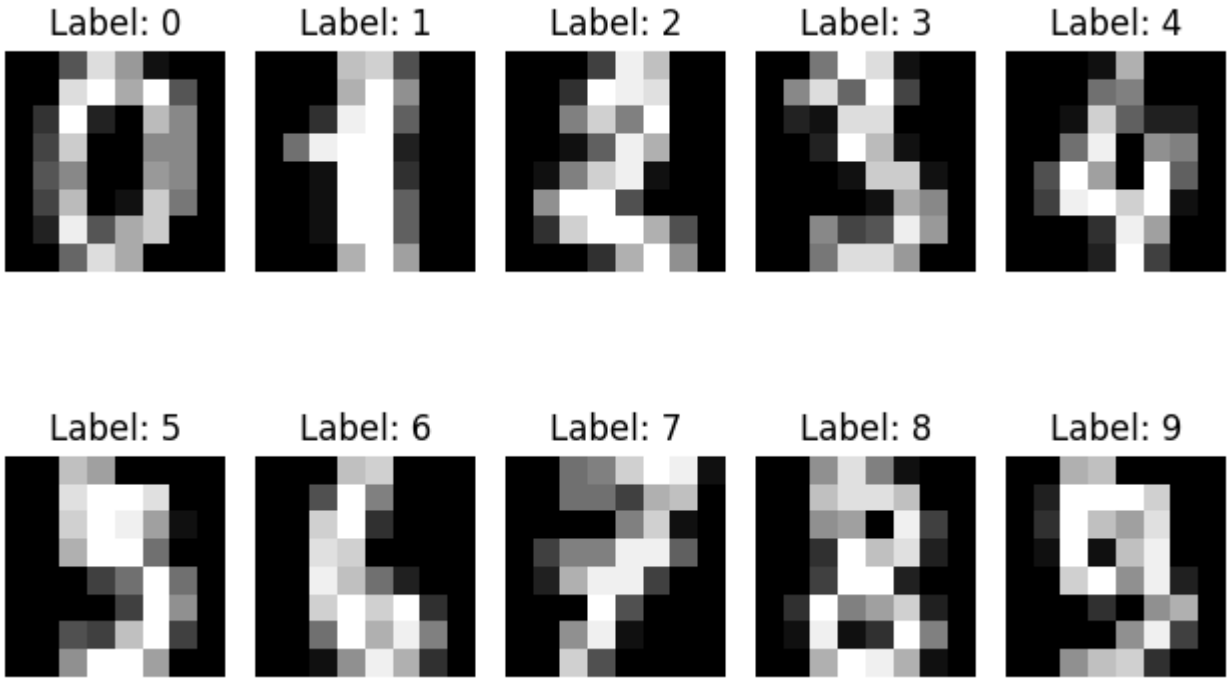
```
# Print to show there are 1797 images (8 by 8)
print("Images Shape" , digits.images.shape)

# Print to show there are 1797 image data
# (8 by 8 images for a dimensionality of 64)
print("Image Data Shape" , digits.data.shape)

# Print to show there are 1797 labels (integers from 0-9)
print("Label Data Shape", digits.target.shape)
```

```
Images Shape (1797, 8, 8)
Image Data Shape (1797, 64)
Label Data Shape (1797,)
```

In [4]:

```
def plot_mnist_sample(digits):
    # Setting the figure size for the plot
    plt.figure()

    # Looping through the first 10 digits to plot
    for i in range(10):
        # Creating a 2x5 grid layout for subplots
        plt.subplot(2, 5, i + 1)

        # Displaying the image for the
        # current digit in grayscale
        plt.imshow(digits.images[i], cmap='gray')

        # Adding a title to each subplot with the
        # corresponding label
        plt.title(f"Label: {digits.target[i]}")

        # Hiding the axes for a cleaner
        # visualization
        plt.axis('off')

    # Adjusting layout to prevent overlapping
    plt.tight_layout()

    # Displaying the plot
    plt.show()
```

In [5]:
```
# PLOT CODE: DO NOT CHANGE
# This code is for you to plot the results
plot_mnist_sample(digits)
```



Ζήτημα 1.3: Αναγνώριση χειρόγραφων ψηφίων με Sklearn [2 μονάδες]

Ένα από τα πιο ενδιαφέροντα πράγματα σχετικά με τη βιβλιοθήκη Sklearn είναι ότι παρέχει έναν εύκολο τρόπο δημιουργίας και κλήσης/χρήσης διαφορετικών μοντέλων. Σε αυτό το μέρος της άσκησης, θα αποκτήσετε εμπειρία με τα μοντέλα ταξινόμησης `LogisticRegressionClassifier` (ταξινόμηση με λογιστική παλινδρόμηση) και `kNNCalssifier` (ταξινόμηση με τη μέθοδο κ-κοντινότερων γειτόνων).

Ακολουθούν αρχικά 2 βοηθητικές ρουτίνες: 1) μια *ρουτίνα δημιουργίας* mini-batches (παρτίδων) δεδομένων *εκπαίδευσης* και *ελέγχου*, αντίστοιχα, 2) μια *ρουτίνα ελέγχου* του εκάστοτε ταξινομητή στις παρτίδες δεδομένων (train/test): α) RandomClassifier(), β) LogisticRegressionClassifier(), γ) kNNClassifier καθώς και των ταξινομητών των ζητημάτων 1.4, 1.5, 1.6 και 2.2, 2.4, 2.5. Στη συνέχεια η συνάρτηση train_test_split() διαχωρίζει το σύνολο δεδομένων σε δεδομένα μάθησης (training set: <X_train, y_train>) και ελέγχου (test

set: <X_test, y_test>).

Ο κώδικας που ακολουθεί στη συνέχεια ορίζει κάποιες συναρτήσεις/μεθόδους για 3 ταξινομητές: 2 για τον RandomClassifier() και 3 μεθόδους για τους ταξινομητές LogisticRegressionClassifier() και kNNClassifier(). Οι 2 τελευταίες κλάσσεις έχουν μια μέθοδο **init** για αρχικοποίηση, μια μέθοδο **train** για την εκπαίδευση του μοντέλου και μια μέθοδο **call** για την πραγματοποίηση προβλέψεων. Πρέπει να συμπληρώσετε τα μέρη κώδικα που λείπουν από τις κλάσεις LogisticRegressionClassifier και kNNClassifier, χρησιμοποιώντας τις υλοποιήσεις `LogisticRegression` και `KNeighborsClassifier` από το Sklearn. Τέλος να συμπληρώσετε τον κώδικα για την αξιολόγηση του `KNeighborsClassifier` ταξινομητή στο σύνολο ελέγχου.

In [6]:
```
# DO NOT CHANGE
#### Some helper functions are given below ####
def DataBatch(data, label, batchsize, shuffle=True):
    """
    This function provides a generator for batches of data that
    yields data (batchsize, 3, 32, 32) and labels (batchsize)
    if shuffle, it will load batches in a random order
    """
    n = data.shape[0]

    if shuffle:
        index = np.random.permutation(n)
    else:
        index = np.arange(n)

    for i in range(int(np.ceil(n/batchsize))):
        inds = index[i*batchsize : min(n,(i+1)*batchsize)]
        yield data[inds], label[inds]


def test(testData, testLabels, classifier):
    """
    Call this function to test the accuracy of a classifier
    """
    batchsize=50
    correct=0.0

    for data,label in DataBatch(testData,testLabels,batchsize,shuffle=False):
        prediction = classifier(data)
```

```
        correct += np.sum(prediction==label)

    return correct/testData.shape[0]*100
```

In [7]:
```
# DO NOT CHANGE
# Split data into 90% train and 10% test subsets
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(
    digits.images.reshape((len(digits.images), -1)), digits.target, test_size=0.1, shuffle=False)
```

In [8]:
```
# Importing Logistic Regression model
# from scikit-learn
from sklearn.linear_model import LogisticRegression

# Importing K-Nearest Neighbors classifier
# from scikit-learn
from sklearn.neighbors import KNeighborsClassifier


class RandomClassifier():
    """
    This is a sample classifier.
    Given an input, it outputs a random class.
    """
    def __init__(self, classes=10):
        # Initializing the RandomClassifier
        # with the number of classes
        self.classes = classes

    def __call__(self, x):
        # Generating random predictions
        # for input data
        return np.random.randint(self.classes, size=x.shape[0])


class LogisticRegressionClassifier():
    def __init__(self, sol='liblinear'):
        """
```

```
        Initialize Logistic Regression model.

        Inputs:
        sol: Solver method that the Logistic Regression model would use for optimization.
        """
        # Creating a Logistic Regression model with
        # specified solver and maximum iterations
        self.model = LogisticRegression(solver=sol, max_iter=1000)

    def train(self, trainData, trainLabels):
        """
        Train your model with image data and corresponding labels.

        Inputs:
        trainData: Training images (N,64)
        trainLabels: Labels (N,)
        """
        # Fitting the Logistic Regression model
        # to the training data and labels
        self.model.fit(trainData, trainLabels)

    def __call__(self, x):
        """
        Predict the trained model on test data.

        Inputs:
        x: Test images (N,64)

        Returns:
        predicted labels (N,)
        """
        # Using the trained model to predict
        # labels for the input data
        return self.model.predict(x)


class kNNClassifier():
    def __init__(self, k=3, algorithm='brute'):
        """
        Initialize KNN model.
```

```
        Inputs:
        k: Number of neighbors involved in voting.
        algorithm: Algorithm used to compute nearest neighbors.
        """
        # Creating a K-Nearest Neighbors model
        # with specified parameters
        self.model = KNeighborsClassifier(n_neighbors=k, algorithm=algorithm)

    def train(self, trainData, trainLabels):
        """
        Train your model with image data and corresponding labels.

        Inputs:
        trainData: Training images (N,64)
        trainLabels: Labels (N,)
        """
        # Fitting the KNN model to the
        # training data and labels
        self.model.fit(trainData, trainLabels)

    def __call__(self, x):
        """
        Predict the trained model on test data.

        Inputs:
        x: Test images (N,64)

        Returns:
        predicted labels (N,)
        """
        # Using the trained model
        # to predict labels for the input data
        return self.model.predict(x)
```

In [9]:
```
# TEST CODE: DO NOT CHANGE
randomClassifierX = RandomClassifier()
print ('Random classifier accuracy: %f' % test(X_test, y_test, randomClassifierX))
```

```
Random classifier accuracy: 7.222222
```

In [10]:
```
# TEST CODE: DO NOT CHANGE
# TEST LogisticRegressionClassifier
lrClassifierX = LogisticRegressionClassifier()
lrClassifierX.train(X_train, y_train)
print ('Logistic Regression Classifier classifier accuracy: %f' % test(X_test, y_test, lrClassifierX))
```

```
Logistic Regression Classifier classifier accuracy: 93.888889
```

In [11]:
```
# TEST kNNClassifier
knnClassifierX = kNNClassifier(k=3)
knnClassifierX.train(X_train, y_train)
print('k-NN Classifier accuracy: %f' % test(X_test, y_test, knnClassifierX))
```

```
k-NN Classifier accuracy: 96.666667
```

Ζήτημα 1.4: Πίνακας Σύγχυσης [2 μονάδες]

Ένας πίνακας σύγχυσης είναι ένας 2Δ πίνακας που χρησιμοποιείται συχνά για να περιγράψει την απόδοση ενός μοντέλου ταξινόμησης σε ένα σύνολο δεδομένων ελέγχου/δοκιμής (test data) για τα οποία είναι γνωστές οι πραγματικές τιμές (known labels). Εδώ θα υλοποιήσετε τη συνάρτηση που υπολογίζει τον πίνακα σύγχυσης για έναν ταξινομητή. Ο πίνακας (M) πρέπει να είναι $n \times n$ όπου n είναι ο αριθμός των κλάσεων/κατηγοριών. Η καταχώριση `M[i,j]` πρέπει να περιέχει το ποσοστό/λόγο των εικόνων της κατηγορίας `i` που ταξινομήθηκε ως κατηγορία `j`. Αν οι καταχωρήσεις `M[i,j]` έχουν υπολογιστεί σωστά, τότε τα στοιχεία `M[k,j]` κατά μήκος μιας γραμμής k για $j \neq k$ (εκτός της κύριας διαγωνίου) αναμένεται να αντιστοιχούν σε "ψευδώς αρνητικές" ταξινομήσεις (false negatives), ενώ τα στοιχεία `M[i,k]` κατά μήκος μιας στήλης k για $i \neq k$ (εκτός της κύριας διαγωνίου) αναμένεται να αντιστοιχούν σε "ψευδώς θετικές" ταξινομήσεις (false positives). Το ακόλουθο παράδειγμα δείχνει τον πίνακα σύγχυσης για τον `RandomClassifier` ταξινομητή. Ο στόχος σας είναι να σχεδιάσετε τα αποτελέσματα για τον `LogisticRegressionClassifier` και τον `kNNClassifier` ταξινομητή. *Να δώσετε προσοχή* στο άθροισμα των στοιχείων μιας γραμμής (false negatives) `M[i,:]` ώστε να αθροίζει σωστά, στο συνολικό ποσοστό ταξινόμησης (100% ή 1). Αν δεν συμβαίνει κάτι τέτοιο, μπορεί να χρειαστείτε κανονικοποίηση των τιμών.

In [12]:
```
# Importing tqdm for creating
# progress bars
from tqdm import tqdm
```

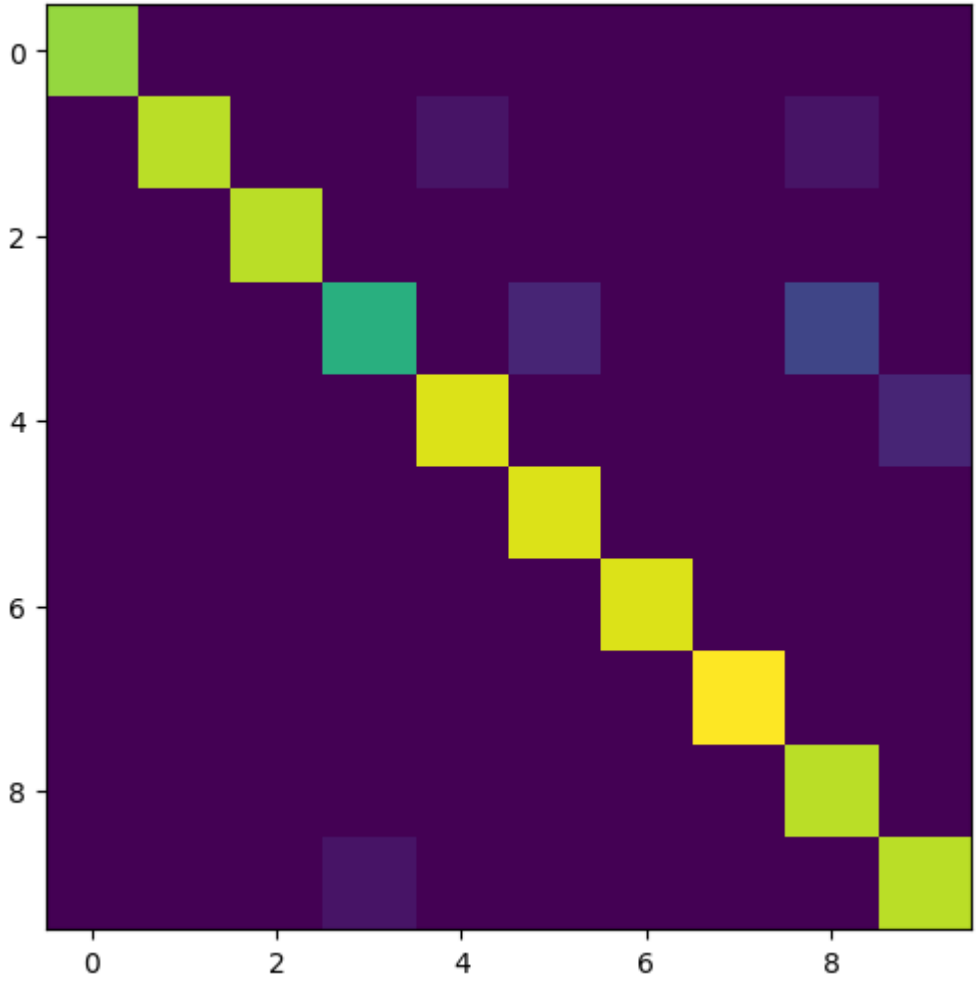
```
def Confusion(testData, testLabels, classifier):
    """
    Compute the confusion matrix and accuracy for the classifier.

    Inputs:
    testData: Test images (N,64)
    testLabels: True labels (N,)
    classifier: The classifier to evaluate

    Returns:
    M: Confusion matrix (10x10)
    acc: Accuracy in percentage
    """
    # Batch size for processing the
    # test data in chunks
    batchsize = 50

    # Counter for correct
    # predictions
    correct = 0

    # Initializing a 10x10
    # confusion matrix
    M = np.zeros((10, 10))

    # Counter for the total
    # number of samples processed
    count = 0

    # Iterating through test data in batches
    for data, label in tqdm(DataBatch(testData, testLabels, batchsize, shuffle=False), total=len(testData) // batch
        # Getting predictions from the
        # classifier for the current batch
        predictions = classifier(data)

        # Counting occurrences of each
        # true-predicted pair
        unique_labels, unique_counts = np.unique(
            np.stack((label, predictions), axis=1), axis=0, return_counts=True
        )
```

```
        # Updating the confusion matrix with
        # the counts of each true-predicted pair
        for (true_class, predicted_class), occurrences in zip(unique_labels, unique_counts):
            M[true_class, predicted_class] += occurrences

        # Counting the number of correct predictions
        correct += np.sum(label == predictions)

        # Updating the total number of processed samples
        count += len(label)

    # Calculating accuracy as a percentage
    acc = correct / count * 100.0

    return M, acc

def VisualizeConfussion(M):
    """
    Visualize the confusion matrix.

    Inputs:
    M: Confusion matrix (10x10)
    """
    # Setting the figure size
    # for the visualization
    plt.figure(figsize=(14, 6))

    # Displaying the confusion
    # matrix as an image
    plt.imshow(M)

    # Showing the plot
    plt.show()

    # Printing the rounded confusion
    # matrix values for reference
    print(np.round(M, 2))
```

In [13]:
```
# TEST/PLOT CODE: DO NOT CHANGE
# TEST LogisticRegressionClassifier
```

```
M,acc = Confusion(X_test, y_test, lrClassifierX)
VisualizeConfussion(M)
```

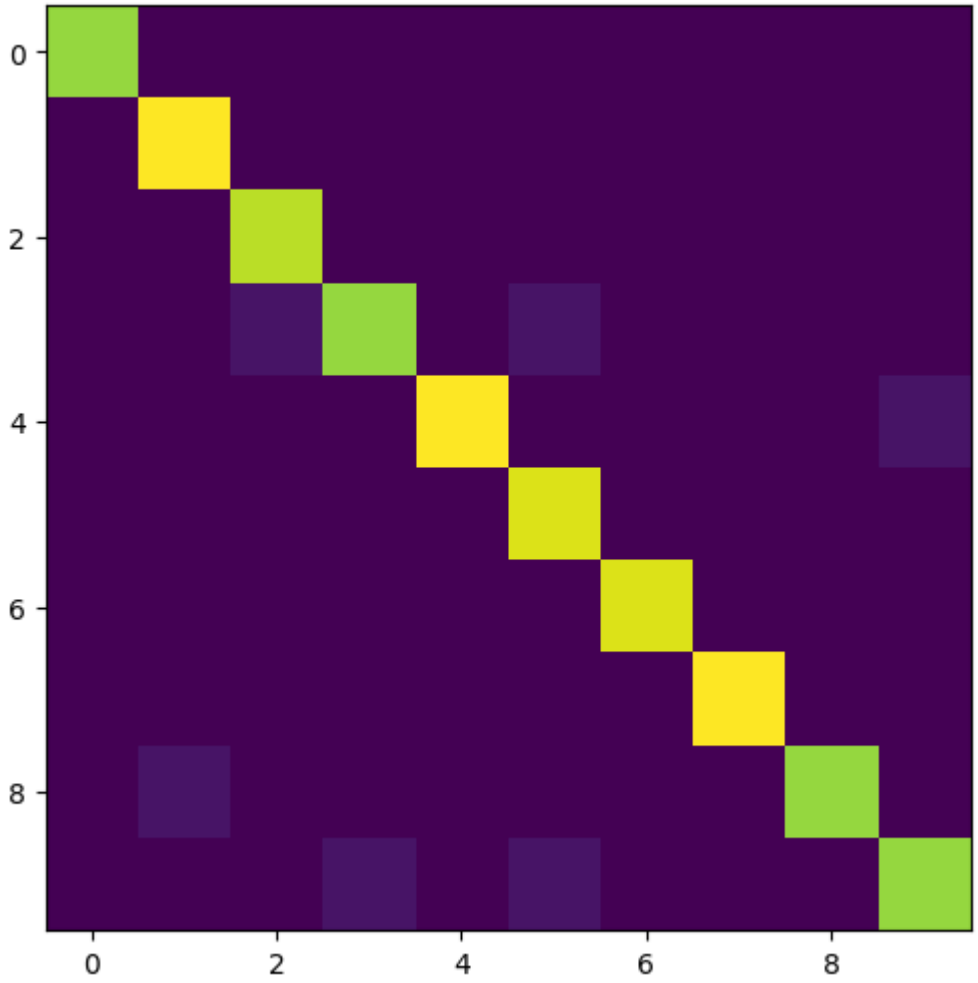
```
4it [00:00, 1052.92it/s]
```



```
[[16.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0. 17.  0.  0.  1.  0.  0.  0.  1.  0.]
 [ 0.  0. 17.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0. 12.  0.  2.  0.  0.  4.  0.]
 [ 0.  0.  0.  0. 18.  0.  0.  0.  0.  2.]
 [ 0.  0.  0.  0.  0. 18.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0. 18.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0. 19.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0. 17.  0.]
 [ 0.  0.  0.  1.  0.  0.  0.  0.  0. 17.]]
```

In [14]:
```
# TEST/PLOT CODE: DO NOT CHANGE
# TEST kNNClassifier
M,acc = Confusion(X_test, y_test, knnClassifierX)
VisualizeConfussion(M)
```

```
4it [00:00, 327.27it/s]
```




```
[[16.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0. 19.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0. 17.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  1. 16.  0.  1.  0.  0.  0.  0.]
 [ 0.  0.  0.  0. 19.  0.  0.  0.  0.  1.]
 [ 0.  0.  0.  0.  0. 18.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0. 18.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0. 19.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.  0.  0. 16.  0.]
 [ 0.  0.  0.  1.  0.  1.  0.  0.  0. 16.]]
```

Ζήτημα 1.5: κ-Κοντινότεροι Γείτονες (k-Nearest Neighbors/kNN) [4 μονάδες]

Για αυτό το πρόβλημα, θα ολοκληρώσετε έναν απλό ταξινομητή kNN χωρίς χρήση του πακέτου Sklearn. Η μέτρηση της απόστασης είναι η Ευκλείδεια απόσταση (L2 norm) στον χώρο των pixel, την οποία και θα πρέπει να υλοποιήσετε (`euclidean_distance`). Το k αναφέρεται στον αριθμό των γειτόνων που συμμετέχουν στην ψηφοφορία για την ομάδα/κλάση. Έπειτα, θα πρέπει να υλοποιήσετε την `find_k_nearest_neighbors` που υπολογίζει την απόσταση του δοθέντος δείγματος από όλα τα δείγματα εκπαίδευσης, ταξινομεί τις αποστάσεις και επιστρέφει τους δείκτες των k κοντινότερων γειτόνων. Τέλος, για κάθε δείγμα του συνόλου δοκιμών, μέσω της ρουτίνας `__call__` βρίσκετε τους k κοντινότερους γείτονες και εκτελείτε "ψηφοφορία" για την προβλεπόμενη κλάση.

In [15]:
```python
class kNNClassifier_v1_5():
    def __init__(self, k=3):
        # Initialize the kNN classifier with
        # he number of neighbors (k)
        self.k = k

    def train(self, trainData, trainLabels):
        """
        Stores the training data.

        Inputs:
        trainData: Training data samples (N, D)
        trainLabels: Labels corresponding to the training data (N,)
        """
        # Storing training data and labels
        # for reference during prediction
        self.X_train = trainData
        self.y_train = trainLabels

    def euclidean_distance(self, x1, x2):
        """
        Calculates the Euclidean distance between two vectors.

        Inputs:
        x1: First vector (D,)
        x2: Second vector (D,)

        Returns:
```

```
        Euclidean distance (float)
        """
        # Calculating the Euclidean distance as the
        # square root of the sum of squared differences
        return np.sqrt(np.sum((x1 - x2) ** 2))

    def find_k_nearest_neighbors(self, x):
        """
        Finds the k nearest neighbors for the given input sample x.

        Inputs:
        x: Input sample (D,)

        Returns:
        indices: Indices of the k nearest neighbors (k,)
        """
        # Calculating distances from x to
        # all training samples
        distances = np.array([self.euclidean_distance(x, x_train) for x_train in self.X_train])

        # Getting the indices of the k
        # smallest distances
        indices = np.argsort(distances)[:self.k]
        return indices

    def __call__(self, X):
        """
        Predicts the labels for the input data X using the kNN classifier.

        Inputs:
        X: Test data samples (N, D)

        Returns:
        predicted_labels: Predicted labels for the test data (N,)
        """
        # Initializing a list to store
        # predicted labels
        predicted_labels = []

        # Iterating through each test sample
        for x in X:
```

```
            # Finding the k nearest neighbors
            # for the current sample
            neighbor_indices = self.find_k_nearest_neighbors(x)

            # Retrieving the labels of
            # the k nearest neighbors
            neighbor_labels = [self.y_train[i] for i in neighbor_indices]

            # Determining the most frequent
            # label among the neighbors using voting
            most_common_label = np.bincount(neighbor_labels).argmax()

            # Appending the predicted label
            # for the current sample
            predicted_labels.append(most_common_label)

        # Returning the predicted labels
        # as a numpy array
        return np.array(predicted_labels)
```

In [16]:
```
# TEST/PLOT CODE: DO NOT CHANGE
# TEST kNNClassifierManual
knnClassifierManualX = kNNClassifier_v1_5()
knnClassifierManualX.train(X_train, y_train)
print ('kNN classifier accuracy: %f' % test(X_test, y_test, knnClassifierManualX))
```

```
kNN classifier accuracy: 96.666667
```

Ζήτημα 1.6: PCA + κ-κοντινότεροι γείτονες (PCA/k-NN) [6 μονάδες]

Σε αυτό το ζήτημα θα εφαρμόσετε έναν απλό ταξινομητή kNN, αλλά στον χώρο PCA, δηλαδή όχι τον χώρο των πίξελ, αλλά αυτόν που προκύπτει μετά από ανάλυση σε πρωτεύουσες συνιστώσες των εικόνων του συνόλου εκπαίδευσης (για k=3 και 25 πρωτεύουσες συνιστώσες).

Θα πρέπει να υλοποιήσετε μόνοι σας την PCA χρησιμοποιώντας "Singular Value Decomposition (SVD)". Η χρήση του `sklearn.decomposition.PCA` ή οποιουδήποτε άλλου πακέτου που υλοποιεί άμεσα μετασχηματισμούς PCA **θα οδηγήσει σε μείωση μονάδων**. Μπορείτε να χρησιμοποιήσετε τη ρουτίνα `np.linalg.eigh` για την υλοποίησή σας. Προσοχή στον χειρισμό μηδενικών `singular values` μέσα στην υλοποίηση της `svd` .

Μπορείτε να χρησιμοποιήσετε την προηγούμενη υλοποίηση του ταξινομητή `kNNClassifier_v1_5` σε αυτό το ζήτημα (Υπόδειξη: ορισμός πεδίου `self.knn = ...` μέσα στην `__init__`). Διαφορετικά, μπορείτε να υλοποιήσετε εκ νέου τον ταξινομητή kNN μέσα στην `__call__` με χρήση της `np.linalg.norm`. Μη ξεχάσετε να καλέσετε την προηγούμενη υλοποίηση του πίνακα σύγχυσης `Confusion` για την αξιολόγηση της μεθόδου στο τέλος του ζητήματος.

Είναι ο χρόνος ελέγχου για τον ταξινομητή PCA-kNN μεγαλύτερος ή μικρότερος από αυτόν για τον ταξινομητή kNN; Εφόσον διαφέρει, **σχολιάστε** γιατί στο τέλος της άσκησης.

In [17]:
```
def svd(A):
    """
    Computes the Singular Value Decomposition (SVD) of matrix A.
    Returns U, singular_values, and V.T.

    Inputs:
    A: Input matrix (M, N)

    Returns:
    U: Left singular vectors (M, M)
    singular_values: Singular values (min(M, N))
    V_T: Right singular vectors transposed (N, N)
    """
    # Performing SVD on the matrix A
    U, singular_values, V_T = np.linalg.svd(A, full_matrices=False)
    return U, singular_values, V_T


class PCAKNNClassifer():
    def __init__(self, components=25, k=3):
        """
        Initializes the PCA-KNN classifier.

        Inputs:
        components: Number of principal components to keep.
        k: Number of neighbors for the kNN classifier.
        """
        # Number of principal components
        self.components = components

        # Number of neighbors for kNN
```

```
        self.k = k

        # Placeholder for the kNN model
        self.knn = None

        # Mean of the training data for normalization
        self.mean = None

        # Matrix of principal components
        self.V = None

    def train(self, trainData, trainLabels):
        """
        Trains the PCA-KNN classifier with the given data and labels.

        Inputs:
        trainData: Training data samples (N, D)
        trainLabels: Training labels (N,)
        """
        # Computing the mean for normalization
        self.mean = np.mean(trainData, axis=0)
        X_centered = trainData - self.mean

        # Performing Singular Value
        # Decomposition (SVD) on the centered data
        U, D, V_T = svd(X_centered)

        # Selecting the top 'components' principal components
        self.V = V_T[:self.components, :]

        # Projecting the data onto the space
        # of the principal components
        Z = np.dot(X_centered, self.V.T)

        # Training the kNN classifier with
        # the projected data
        self.knn = kNNClassifier_v1_5(k=self.k)
        self.knn.train(Z, trainLabels)

    def __call__(self, X):
        """
```

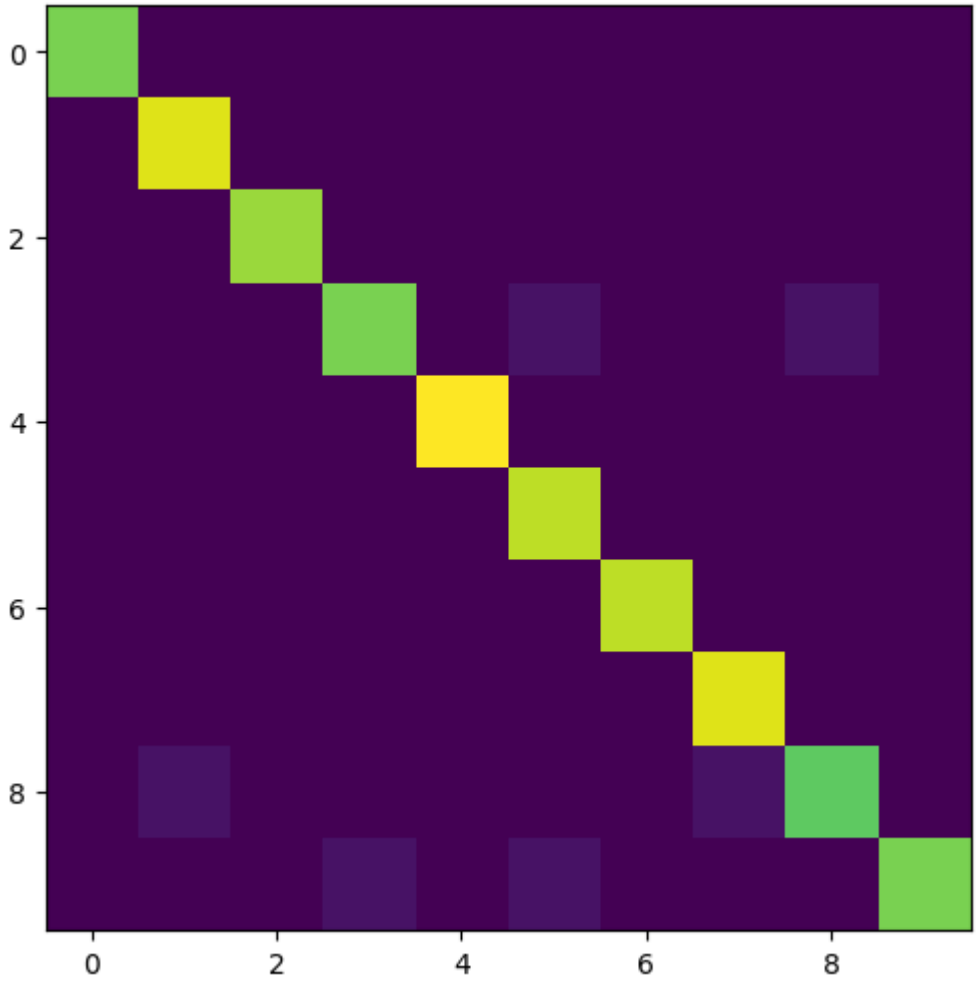
```
        Makes predictions on new data using the trained PCA-KNN classifier.

        Inputs:
        X: Test data samples (M, D)

        Returns:
        Predicted labels for the test data (M,)
        """
        # Normalizing the test data using
        # the mean of the training data
        X_centered = X - self.mean

        # Projecting the test data onto
        # the space of the principal components
        Z = np.dot(X_centered, self.V.T)

        # Making predictions using
        # the kNN classifier
        return self.knn(Z)


# Test the PCA-KNN classifier using the first
# 100 training examples (useful for debugging)
pcaknnClassiferX = PCAKNNClassifer()
pcaknnClassiferX.train(X_train[:100], y_train[:100])

# Testing and printing the accuracy of the
# PCA-KNN classifier
print('PCA-kNN classifier accuracy: %f' % test(X_test, y_test, pcaknnClassiferX))
```

```
PCA-kNN classifier accuracy: 85.000000
```

In [18]:
```
# Initializing the PCA-KNN classifier with
# default parameters (25 principal components and k=3)
pcaknnClassifer = PCAKNNClassifer()

# Training the PCA-KNN classifier on
# the full training dataset
pcaknnClassifer.train(X_train, y_train)

# Generating the confusion matrix and
```

```
# calculating accuracy for the PCA-KNN
# classifier on the test dataset
M_pca, accuracy_pca = Confusion(X_test, y_test, pcaknnClassifer)

# Visualizing the confusion matrix
VisualizeConfussion(M_pca)
```

```
4it [00:01,  2.70it/s]
```



```
[[16.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0. 19.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0. 17.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0. 16.  0.  1.  0.  0.  1.  0.]
 [ 0.  0.  0.  0. 20.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0. 18.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0. 18.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0. 19.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.  0.  1. 15.  0.]
 [ 0.  0.  0.  1.  0.  1.  0.  0.  0. 16.]]
```

- Σχολιασμός του χρόνου εκτέλεσης PCA-kNN σε σχέση με τον kNN.

Ο PCA-kNN απαιτεί περισσότερο χρόνο προεπεξεργασίας λόγω της μείωσης διαστάσεων (PCA), αλλά μειώνει το χρόνο πρόβλεψης του kNN για μεγάλα δεδομένα. Αυτό οδηγεί σε ένα συμβιβασμό μεταξύ αρχικού κόστους και αποδοτικότητας πρόβλεψης.

Άσκηση 2: Βαθιά Μάθηση [15 μονάδες + bonus 5 μονάδες (ζήτημα 2.5)]

Ζήτημα 2.1 Αρχική Εγκατάσταση (απεικόνιση παραδειγμάτων) [1 μονάδα]

- **Τοπικά** (jupyter): Ακολουθήστε τις οδηγίες στη διεύθυνση https://pytorch.org/get-started/locally/ για να εγκαταστήσετε την PyTorch τοπικά στον υπολογιστή σας. Για παράδειγμα, αφού δημιουργήσετε και ενεργοποιήσετε κάποιο εικονικό περιβάλλον anaconda με τις εντολές: π.χ. `(base)$ conda create -n askisi` , `(base)$ conda activate askisi` , η εντολή `(askisi)$ conda install pytorch torchvision torchaudio cpuonly -c pytorch` εγκαθιστά την βιβλιοθήκη "PyTorch" σε περιβάλλον Linux/Windows χωρίς GPU υποστήριξη.

Προσοχή σε αυτό το σημείο, αν τρέχετε την άσκηση τοπικά σε jupyter, εκτός της εγκατάστασης του PyTorch, θα χρειαστούν ξανά και κάποιες βιβλιοθήκες `matplotlib` , `scipy` , `tqdm` και `sklearn` (όπως και στην 1η άσκηση), μέσα στο περιβάλλον 'askisi', πριν ανοίξετε το jupyter: `(askisi)$ conda install matplotlib tqdm scipy` και `(askisi)$ conda install -c anaconda scikit-learn` . Αυτό χρειάζεται διότι σε ορισμένες περιπτώσεις, αφού εγκαταστήσετε τις βιβλιοθήκες που απαιτούνται, πρέπει να εξασφαλίσετε ότι ο *Python Kernel* αναγνωρίζει την προϋπάρχουσα εγκατάσταση (PyTorch, matplotlib, tqdm, κτλ.). Τέλος, χρειάζεται να εγκαταστήσετε το jupyter ή jupyterlab μέσω του περιβάλλοντος conda: `(askisi)$ conda install jupyter` και μετά να εκτελέσετε `(askisi)$ jupyter notebook` για να ανοίξετε το jupyter με τη σωστή εγκατάσταση. Αν όλα έχουν γίνει σωστά, θα πρέπει ο *Python*

Kernel να βλέπει όλα τα 'modules' που χρειάζεστε στη 2η άσκηση. Διαφορετικά, μπορείτε να εγκαταστήσετε εξ' αρχής όλες τις βιβλιοθήκες, από την αρχή υλοποίησης της εργασίας, μέσα στο εικονικό περιβάλλον *askisi* ώστε να μην είναι απαραίτητη εκ νέου η εγκατάσταση των βιβλιοθηκών που θα χρειαστούν στη 2η άσκηση.

- **Colab**: **Αν** χρησιμοποιείτε google colab, τότε δεν θα χρειαστεί λογικά κάποιο βήμα εγκατάστασης. Αν ωστόσο σας παρουσιαστεί κάποιο πρόβλημα με απουσία πακέτου, π.χ. "ModuleNotFoundError - torchvision", τότε μπορείτε απλώς να το εγκαταστήσετε με χρήση του εργαλείου `pip` εκτελώντας την αντίστοιχη εντολή (π.χ. "!pip install torchvision") σε ένα νέο κελί του notebook.

Σημείωση: Δεν θα είναι απαραίτητη η χρήση GPU για αυτήν την άσκηση, γι' αυτό μην ανησυχείτε αν δεν έχετε ρυθμίσει την εγκατάσταση με υποστήριξη GPU. Επιπλέον, η εγκατάσταση με υποστήριξη GPU είναι συχνά πιο δύσκολη στη διαμόρφωση, γι' αυτό και προτείνεται να εγκαταστήσετε μόνο την έκδοση CPU.

Εκτελέστε τις παρακάτω εντολές για να επαληθεύσετε την εγκατάστασή σας (PyTorch).

In [19]:
```
# Importing the SciPy library
# for scientific computations
import scipy

# Importing the neural network
# module from PyTorch
import torch.nn as nn

# Importing functional utilities
# for neural networks in PyTorch
import torch.nn.functional as F

# Importing PyTorch for tensor
# computations and deep learning
import torch

# Importing Variable for wrapping
# tensors for gradient computation
from torch.autograd import Variable


# Creating a 5x3 tensor filled
# with random values
x = torch.rand(5, 3)
```

```
# Printing the tensor to
# display its random values
print(x)
```

```
tensor([[0.7117, 0.6265, 0.1357],
        [0.9233, 0.4282, 0.3284],
        [0.4929, 0.7422, 0.7441],
        [0.1626, 0.5587, 0.9198],
        [0.3632, 0.1305, 0.4415]])
```

Σε αυτή την άσκηση, θα χρησιμοποιήσουμε το πλήρες σύνολο δεδομένων της βάσης δεδομένων MNIST με τις εικόνες ψηφίων 28x28 pixel (60.000 εικόνες εκπαίδευσης, 10.000 εικόνες ελέγχου).

Ο κώδικας που ακολουθεί "κατεβάζει" το σύνολο δεδομένων MNIST της κλάσης torchvision.datasets, στο φάκελο `mnist` (του root καταλόγου). Μπορείτε να αλλάξετε τον κατάλογο που δείχνει η μεταβλητή `path` στη διαδρομή που επιθυμείτε. Ενδεικτικό `path` σε περιβάλλον Windows: **path = 'C:/Users/user/Υπολογιστική Όραση/assignments/assignment/'**. Στην περίπτωση που εργάζεστε μέσω **colab** μπορεί να χρειαστεί η φόρτωση του καταλόγου στο drive, εκτελώντας `from google.colab import drive` και `drive.mount('/content/gdrive')` και μετά θέτοντας π.χ το **path = '/content/gdrive/assignment/'**.

- Θα πρέπει να απεικονίσετε σε ένα σχήμα 2x5 ένα τυχαίο παράδειγμα εικόνας που αντιστοιχεί σε κάθε ετικέτα (κατηγορία) από τα δεδομένα εκπαίδευσης (αντίστοιχα του ζητήματος 1.2).

In [20]:
```
# Importing PyTorch for
# tensor computations
import torch

# Importing torchvision for
# accessing datasets
import torchvision.datasets as datasets

# Importing additional libraries
# for visualization and numerical operations
import matplotlib.pyplot as plt
import numpy as np


# Define the directory path where
# the MNIST dataset will be downloaded/stored
```

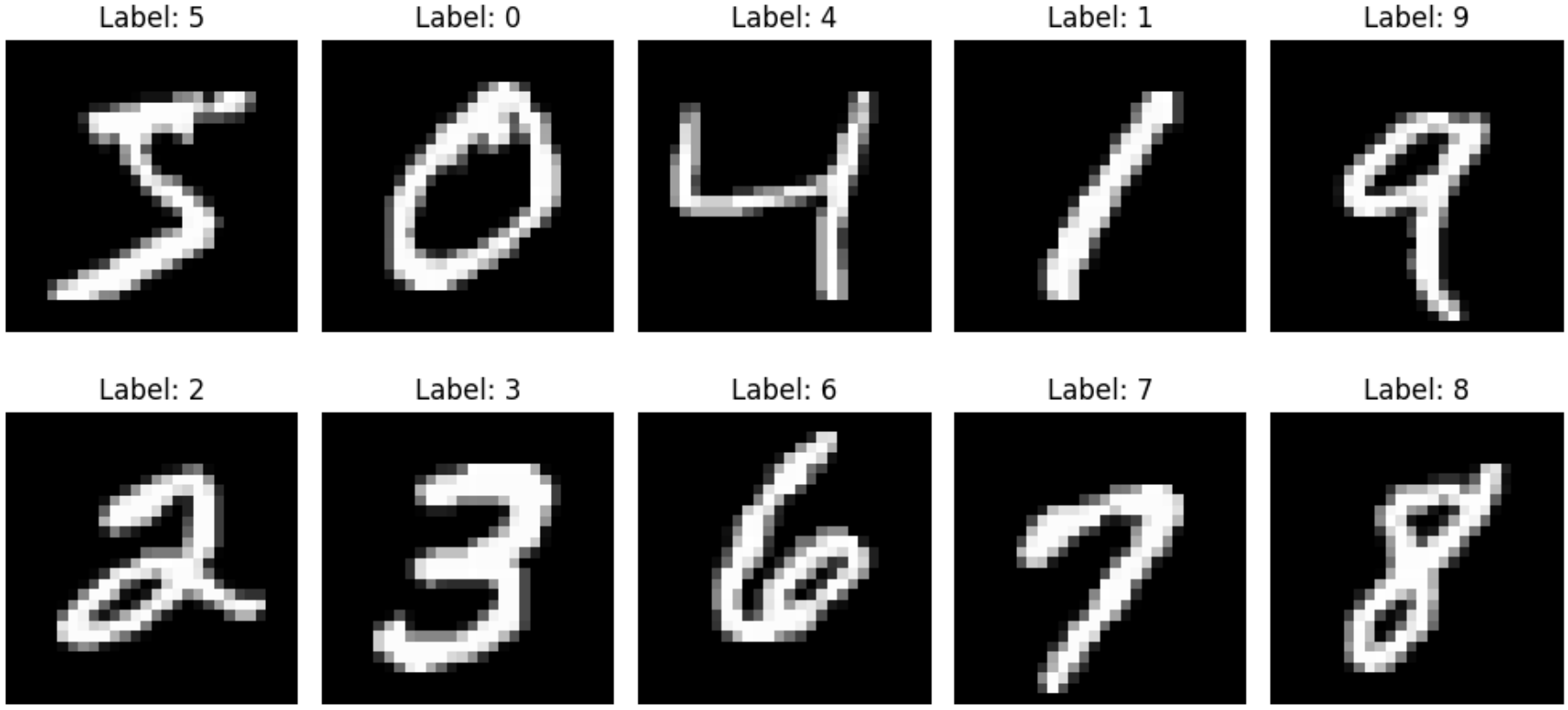
```
path = './mnist/'

# Load the MNIST training dataset
train_dataset = datasets.MNIST(root=path, train=True, download=True)

# Extract the images (X_train) and corresponding
# labels (y_train) from the training dataset
# Convert training images to NumPy array
X_train = train_dataset.data.numpy()

# Convert training labels to NumPy array
y_train = train_dataset.targets.numpy()

# Load the MNIST testing dataset
test_dataset = datasets.MNIST(root=path, train=False, download=True)

# Extract the images (X_test) and corresponding
# labels (y_test) from the testing dataset
# Convert testing images to NumPy array
X_test = test_dataset.data.numpy()

# Convert testing labels to NumPy array
y_test = test_dataset.targets.numpy()
```

In [21]:

```
def plot_mnist_sample_high_res(X_train, y_train):
    """
    This function plots a sample image for each category (0-9).
    The result is a figure with a 2x5 grid of images.

    Inputs:
    X_train: Array of training images (N, H, W)
    y_train: Array of training labels (N,)
    """
    # Create a dictionary to store one example image per label
    label_to_image = {}

    for i, label in enumerate(y_train):
        # Add the first occurrence of each
        # label to the dictionary
        if label not in label_to_image:
```

```
            label_to_image[label] = X_train[i]

        # Stop once all labels (0-9)
        # have been added
        if len(label_to_image) == 10:
            break

    # Create a 2x5 grid to display the images
    fig, axes = plt.subplots(2, 5, figsize=(10, 5))

    for ax, (label, image) in zip(axes.flatten(), label_to_image.items()):
        # Plot each image in grayscale
        ax.imshow(image.squeeze(), cmap='gray')

        # Set the title for each subplot
        # to indicate the label
        ax.set_title(f"Label: {label}")

        # Hide the axes for a cleaner
        # visualization
        ax.axis('off')

    # Adjust layout to prevent
    # overlapping of titles and images
    plt.tight_layout()

    # Display the plot
    plt.show()
```

In [22]:
```
# PLOT CODE: DO NOT CHANGE
# This code is for you to plot the results.
plot_mnist_sample_high_res(X_train, y_train)
```



Ζήτημα 2.2: Εκπαίδευση Νευρωνικού Δικτύου με PyTorch [5 μονάδες]

Ακολουθεί ένα τμήμα βοηθητικού κώδικα για την εκπαίδευση των βαθιών νευρωνικών δικτύων (Deep Neural Networks - DNN).

- Ολοκληρώστε τη συνάρτηση train_net() για το παρακάτω DNN.

Θα πρέπει να συμπεριλάβετε τις λειτουργίες της διαδικασίας της εκπαίδευσης σε αυτή τη συνάρτηση. Αυτό σημαίνει ότι για μια παρτίδα/υποσύνολο δεδομένων (batch μεγέθους 50) πρέπει να αρχικοποιήσετε τις παραγώγους, να υλοποιήσετε τη διάδοση προς τα εμπρός της πληροφορίας (forward propagation), να υπολογίσετε το σφάλμα εκτίμησης, να κάνετε οπισθοδιάδοση της πληροφορίας (μετάδοση προς τα πίσω των παραγώγων σφάλματος ως προς τα βάρη - backward propagation), και τέλος, να ενημερώσετε τις παραμέτρους (weight update). Θα πρέπει να επιλέξετε μια κατάλληλη συνάρτηση απώλειας και βελτιστοποιητή (optimizer) από την βιβλιοθήκη PyTorch για αυτό το πρόβλημα.

Αυτή η συνάρτηση θα χρησιμοποιηθεί στα επόμενα ζητήματα με διαφορετικά δίκτυα. Θα μπορείτε δηλαδή να χρησιμοποιήσετε τη μέθοδο `train_net` για να εκπαιδεύσετε το βαθύ νευρωνικό σας δίκτυο, εφόσον προσδιορίσετε τη συγκεκριμένη αρχιτεκτονική σας και εφαρμόσετε το `forward pass` σε μια υπο/κλάσση της DNN (βλ. παράδειγμα "LinearClassifier(DNN)"). Μπορείτε να ανατρέξετε στη διεύθυνση https://pytorch.org/tutorials/beginner/pytorch_with_examples.html για περισσότερες πληροφορίες. Επίσης, ένα αρκετά χρήσιμο "tutorial" περιλαμβάνεται στο σημειωματάριο jupyter `(tutorial1_pytorch_introduction.ipynb)` στο φάκελο της αναρτημένης εργασίας στη σελίδα ecourse του μαθήματος.

Στο τέλος, μπορείτε να χρησιμοποιήσετε την υφιστάμενη υλοποίηση από την 1η άσκηση για την αξιολόγηση της απόδοσης (`Confusion` και `VisualizeConfussion`).

In [23]:
```
# Importing PyTorch initialization utilities
import torch.nn.init

# Importing optimizers from PyTorch
import torch.optim as optim

# Importing Variable for wrapping tensors for gradient computations
from torch.autograd import Variable

# Importing Parameter class to define learnable parameters in models
from torch.nn.parameter import Parameter

# Importing tqdm for progress bar visualization
from tqdm import tqdm

# Importing truncated normal distribution for parameter initialization
from scipy.stats import truncnorm


class DNN(nn.Module):
    """
    A basic Deep Neural Network (DNN) implementation.
    """
    def __init__(self):
        # Initializing the base nn.Module class
        super(DNN, self).__init__()
        pass
```

```
    def forward(self, x):
        """
        Forward pass for the network.
        To be implemented in subclasses.
        """
        raise NotImplementedError

    def train_net(self, X_train, y_train, epochs=1, batchSize=50):
        """
        Train the DNN on the provided training data.

        Inputs:
        X_train: Training data samples (N, 28, 28)
        y_train: Training labels (N,)
        epochs: Number of epochs for training
        batchSize: Size of each batch
        """
        # Convert X_train and y_train to
        # torch.Tensor if they are not already
        if not isinstance(X_train, torch.Tensor):
            X_train = torch.tensor(X_train, dtype=torch.float32)
        if not isinstance(y_train, torch.Tensor):
            y_train = torch.tensor(y_train, dtype=torch.long)

        # Defining the loss criterion
        criterion = torch.nn.CrossEntropyLoss()

        # Create parameters if they do not exist
        if not hasattr(self, 'weight1'):
            self.weight1 = torch.nn.Parameter(torch.randn(10, 28 * 28, requires_grad=True))
        if not hasattr(self, 'bias1'):
            self.bias1 = torch.nn.Parameter(torch.randn(10, requires_grad=True))

        # Ensure parameters are registered with the model
        self.register_parameter('weight1', self.weight1)
        self.register_parameter('bias1', self.bias1)

        # Creating the optimizer for the parameters
        optimizer = torch.optim.SGD([self.weight1, self.bias1], lr=0.01, momentum=0.9)

        # Training loop for the specified number of epochs
```

```
        for epoch in range(epochs):
            # Shuffle the dataset indices
            permutation = torch.randperm(X_train.size(0))

            # Process data in batches
            for i in range(0, X_train.size(0), batchSize):
                indices = permutation[i:i + batchSize]
                batch_X, batch_y = X_train[indices], y_train[indices]

                # Convert batch_X and batch_y to torch.Tensor if not already
                if not isinstance(batch_X, torch.Tensor):
                    batch_X = torch.tensor(batch_X, dtype=torch.float32)
                if not isinstance(batch_y, torch.Tensor):
                    batch_y = torch.tensor(batch_y, dtype=torch.long)

                # Flatten the images into vectors
                batch_X = batch_X.view(-1, 28 * 28)
                optimizer.zero_grad()

                # Compute predictions
                outputs = torch.addmm(self.bias1, batch_X, self.weight1.t())
                loss = criterion(outputs, batch_y)

                # Backpropagation and optimization step
                loss.backward()
                optimizer.step()

            print(f"Epoch [{epoch + 1}/{epochs}], Loss: {loss.item():.4f}")

    def __call__(self, x):
        """
        Perform predictions on the given input data.

        Inputs:
        x: Input data samples (N, 28, 28)

        Returns:
        Predicted labels (N,)
        """
        inputs = Variable(torch.FloatTensor(x))
        prediction = self.forward(inputs)
```

```
        return np.argmax(prediction.data.cpu().numpy(), 1)

def weight_variable(shape):
    """
    Initializes a weight variable using a truncated normal distribution.

    Inputs:
    shape: Shape of the weight tensor.

    Returns:
    Parameter: Learnable weight parameter.
    """
    initial = torch.Tensor(truncnorm.rvs(-1/0.01, 1/0.01, scale=0.01, size=shape))
    return Parameter(initial, requires_grad=True)

def bias_variable(shape):
    """
    Initializes a bias variable with a constant value.

    Inputs:
    shape: Shape of the bias tensor.

    Returns:
    Parameter: Learnable bias parameter.
    """
    initial = torch.Tensor(np.ones(shape) * 0.1)
    return Parameter(initial, requires_grad=True)
```

In [24]:

```
class LinearClassifier(DNN):
    """
    A linear classifier that extends the DNN class.
    This model performs a linear transformation followed by prediction.
    """
    def __init__(self, in_features=28*28, classes=10):
        """
        Initializes the Linear Classifier.

        Inputs:
        in_features: Number of input features (default: 28*28 for MNIST images).
        classes: Number of output classes (default: 10 for MNIST digits).
```

```
        """
        # Initialize the base DNN class
        super(LinearClassifier, self).__init__()

        # Define model parameters for weights
        # and biases. Initialize weights
        self.weight1 = weight_variable((classes, in_features))

        # Initialize biases
        self.bias1 = bias_variable((classes))

        # Register the parameters with the model
        # to ensure they are learnable
        self.register_parameter('weight1', self.weight1)
        self.register_parameter('bias1', self.bias1)

    def forward(self, x):
        """
        Performs the forward pass for the linear classifier.

        Inputs:
        x: Input tensor (N, 28, 28)

        Returns:
        y_pred: Predicted logits for each class (N, classes)
        """
        # Perform a linear transformation: y = xW^T + b
        y_pred = torch.addmm(self.bias1, x.view(list(x.size())[0], -1), self.weight1.t())
        return y_pred


# X_train=np.float32(np.expand_dims(X_train,-1))/255
# X_train=X_train.transpose((0,3,1,2))

# X_test=np.float32(np.expand_dims(X_test,-1))/255
# X_test=X_test.transpose((0,3,1,2))

# In case abovementioned 4 lines return error: Modify the lines for transposing X_train
# and X_test by uncommenting the following 4 lines and place the 4 lines above in comments
X_train = np.float32(X_train) / 255.0
X_train = X_train.reshape(-1, 1, 28, 28)
```

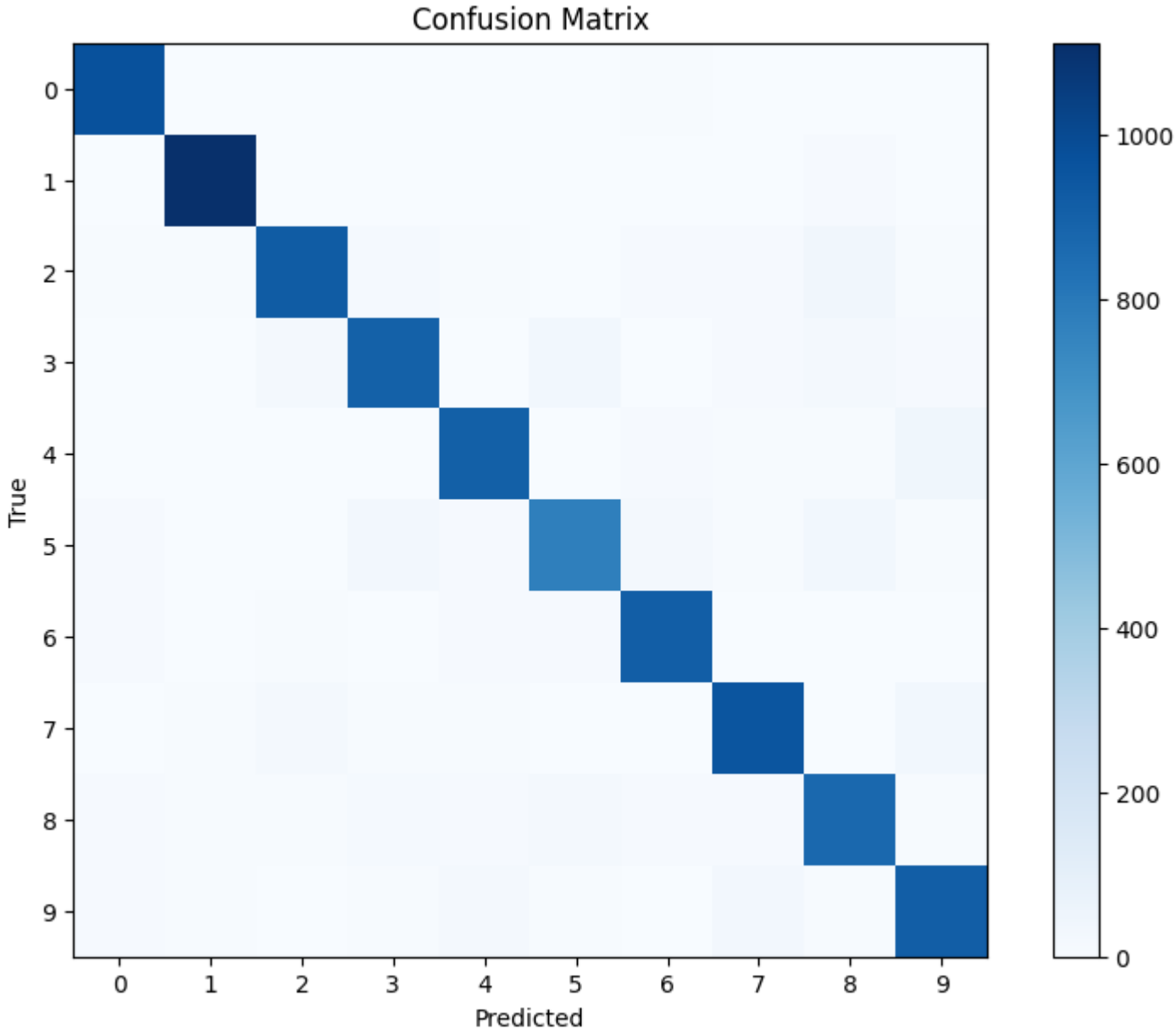
```
X_test = np.float32(X_test) / 255.0
X_test = X_test.reshape(-1, 1, 28, 28)
```

In [25]:
```
# test the example linear classifier (note you should get around 90% accuracy
# for 10 epochs and batchsize 50)
linearClassifier = LinearClassifier()
linearClassifier.train_net(X_train, y_train, epochs=10)
print('Linear classifier accuracy: %f' % test(X_test, y_test, linearClassifier))
```

```
Epoch [1/10], Loss: 0.2634
Epoch [2/10], Loss: 0.3290
Epoch [3/10], Loss: 0.1040
Epoch [4/10], Loss: 0.3135
Epoch [5/10], Loss: 0.1860
Epoch [6/10], Loss: 0.2961
Epoch [7/10], Loss: 0.4980
Epoch [8/10], Loss: 0.2287
Epoch [9/10], Loss: 0.1373
Epoch [10/10], Loss: 0.3640
Linear classifier accuracy: 92.260000
```

In [26]:
```
# Importing confusion matrix computation
# utility from scikit-learn
from sklearn.metrics import confusion_matrix

# Importing for visualization
import matplotlib.pyplot as plt


# Predicting labels for the test dataset
# using the linear classifier
y_pred = linearClassifier(X_test)

# Computing the confusion matrix by comparing
# true and predicted labels
conf_matrix = confusion_matrix(y_test, y_pred)

# Plotting the confusion matrix
# Setting figure size
plt.figure(figsize=(10, 7))
```

```
# Displaying the confusion matrix
# with a color map
plt.imshow(conf_matrix, interpolation='nearest', cmap='Blues')

# Adding a color bar to indicate scale
plt.colorbar()

# Setting x-axis and y-axis ticks to
# represent class labels. Predicted labels
# on the x-axis
plt.xticks(range(len(conf_matrix)), labels=range(len(conf_matrix)))

# True labels on the y-axis
plt.yticks(range(len(conf_matrix)), labels=range(len(conf_matrix)))

# Adding axis labels and a title
# for the confusion matrix
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")

# Displaying the confusion
# matrix plot
plt.show()
```



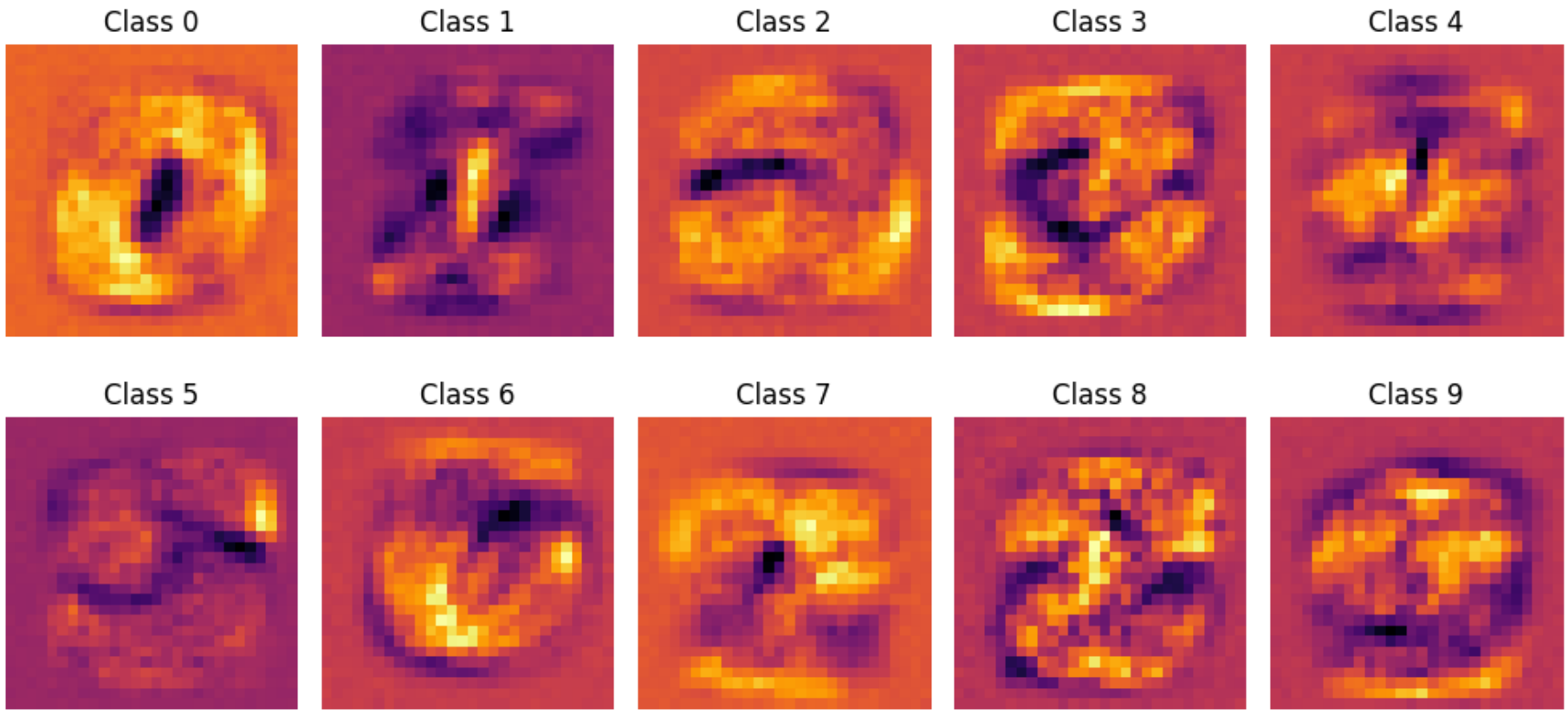
Ζήτημα 2.3: Οπτικοποίηση Βαρών (Visualizing Weights of Single Layer Perceptron) [3 μονάδες]

Αυτός ο απλός γραμμικός ταξινομητής που υλοποιείται στο παραπάνω κελί (το μοντέλο απλά επιστρέφει ένα γραμμικό συνδυασμό της εισόδου) παρουσιάζει ήδη αρκετά καλά αποτελέσματα.

- Σχεδιάστε τα βάρη του φίλτρου που αντιστοιχούν σε κάθε κατηγορία εξόδου (τα **βάρη**/weights, όχι τους όρους *bias*) ως εικόνες. Κανονικοποιήστε τα βάρη ώστε να βρίσκονται μεταξύ 0 και 1 ($z_i = \frac{w_i - min(w)}{max(w) - min(w)}$). Χρησιμοποιήστε έγχρωμους χάρτες όπως "inferno" ή "plasma" για καλά αποτελέσματα (π.χ. cmap='inferno', ως όρισμα της imshow()).
- Σχολιάστε με τι μοιάζουν τα βάρη και γιατί μπορεί να συμβαίνει αυτό.

In [27]:
```
# Extract filter weights corresponding to each class
# from the first layer of the LinearClassifier. Convert
# weights to NumPy array
weights = linearClassifier.weight1.data.cpu().numpy()

# Determine the number of
# classes (rows of the weight matrix)
num_classes = weights.shape[0]

# Create a figure for displaying
# the filter weights
plt.figure(figsize=(10, 5))

# Loop through each class and
# plot the corresponding filter weights
for i in range(num_classes):
    # Reshape the weights for the
    # current class to match image
    # dimensions (28x28)
    w = weights[i].reshape(28, 28)

    # Normalize the weights to a
    # range of [0, 1] for better
    # visualization
    w_norm = (w - w.min()) / (w.max() - w.min())

    # Create a subplot for the
    # current class filter
    plt.subplot(2, 5, i + 1)
```

```
    # Display the normalized weights
    # as an image using the 'inferno' colormap
    plt.imshow(w_norm, cmap='inferno')

    # Set the title of the subplot
    # to indicate the class
    plt.title(f"Class {i}")

    # Hide the axes for a
    # cleaner visualization
    plt.axis('off')

# Adjust layout to prevent
# overlapping of titles and images
plt.tight_layout()

# Display the filter
# weight plots
plt.show()
```



Σχολιασμός των βαρών

Τα βάρη αποκαλύπτουν τα χαρακτηριστικά που το μοντέλο χρησιμοποιεί για να αναγνωρίσει κάθε κατηγορία. Για παράδειγμα, στην κατηγορία "0" εμφανίζεται ένας κύκλος, ενώ στην "1" φαίνεται μια ευθεία γραμμή. Αυτά τα μοτίβα δείχνουν πώς το μοντέλο διακρίνει τα ψηφία μεταξύ τους.

Ζήτημα 2.4: Νευρωνικό δίκτυο πολλαπλών επιπέδων - Multi Layer Perceptron (MLP) [6 μονάδες]

Θα υλοποιήσετε ένα MLP νευρωνικό δίκτυο. Το MLP θα πρέπει να αποτελείται από 2 επίπεδα (πολλαπλασιασμός βάρους και μετατόπιση μεροληψίας/bias - γραμμικός συνδυασμός εισόδου) που απεικονίζονται (map) στις ακόλουθες διαστάσεις χαρακτηριστικών:

- 28x28 -> hidden (50)
- hidden (50) -> classes
- Το κρυμμένο επίπεδο πρέπει να ακολουθείται από μια μη γραμμική συνάρτηση ενεργοποίησης ReLU. Το τελευταίο επίπεδο δεν θα πρέπει να έχει εφαρμογή μη γραμμικής απεικόνισης καθώς επιθυμούμε την έξοδο ακατέργαστων 'logits' (στη μηχανική μάθηση, τα logits είναι οι τιμές που παράγονται από το τελικό επίπεδο ενός μοντέλου πριν περάσουν από μια συνάρτηση ενεργοποίησης softmax. Αντιπροσωπεύουν τις προβλέψεις του μοντέλου για κάθε κατηγορία χωρίς να μετατρέπονται σε πιθανότητες).
- Η τελική έξοδος του υπολογιστικού γράφου (μοντέλου) θα πρέπει να αποθηκευτεί στο self.y καθώς θα χρησιμοποιηθεί στην εκπαίδευση.
- Θα πρέπει να χρησιμοποιήσετε τις helper ρουτίνες `weight_variable` και `bias_variable` στην υλοποίησή σας.

Εμφανίστε τον πίνακα σύγχυσης (confusion matrix - υλοποίηση 1ης άσκησης) και την ακρίβεια (accuracy) μετά την εκπαίδευση. Σημείωση: Θα πρέπει να έχετε ~95-97% ακρίβεια για 10 εποχές (epochs) και μέγεθος παρτίδας (batch size) 50.

Απεικονίστε τα βάρη του φίλτρου που αντιστοιχούν στην αντιστοίχιση από τις εισόδους στις πρώτες 10 εξόδους του κρυμμένου επιπέδου (από τις 50 συνολικά). Μοιάζουν τα βάρη αυτά καθόλου με τα βάρη που απεικονίστηκαν στο προηγούμενο ζήτημα; Γιατί ή γιατί όχι?

Αναμένεται ότι το μοντέλο εκπαίδευσης θα διαρκέσει από 1 έως μερικά λεπτά για να τρέξει, ανάλογα με τις δυνατότητες της CPU.

In [28]:
```
class MLPClassifer(DNN):
    """
    A Multi-Layer Perceptron (MLP) classifier that extends the DNN class.
    This model consists of an input layer, one hidden layer, and an output layer.
    """
    def __init__(self, in_features=28*28, classes=10, hidden=50):
        """
        Initializes the MLP Classifier.

        Inputs:
        in_features: Number of input features (default: 28*28 for MNIST images).
        classes: Number of output classes (default: 10 for MNIST digits).
        hidden: Number of hidden units in the hidden layer (default: 50).
        """
```

```
        # Initialize the base DNN class
        super(MLPClassifer, self).__init__()

        # Define model parameters for the
        # input-to-hidden layer. Weights
        # for the hidden layer
        self.MLPweight1 = weight_variable((hidden, in_features))

        # Biases for the hidden layer
        self.MLPbias1 = bias_variable((hidden))

        # Define model parameters for the
        # hidden-to-output layer. Weights
        # for the output layer
        self.MLPweight2 = weight_variable((classes, hidden))

        # Biases for the output layer
        self.MLPbias2 = bias_variable((classes))

    def forward(self, x):
        """
        Performs the forward pass for the MLP Classifier.

        Inputs:
        x: Input tensor (N, 28, 28)

        Returns:
        y_pred: Predicted logits for each class (N, classes)
        """
        # Flatten the input tensor to a
        # 2D array (batch size, features)
        x_flat = x.view(x.size(0), -1)

        # Compute the hidden layer activations
        # using ReLU activation function
        hidden_output = torch.relu(torch.addmm(self.MLPbias1, x_flat, self.MLPweight1.t()))

        # Compute the output layer (logits)
        # from the hidden layer
        y_pred = torch.addmm(self.MLPbias2, hidden_output, self.MLPweight2.t())
```

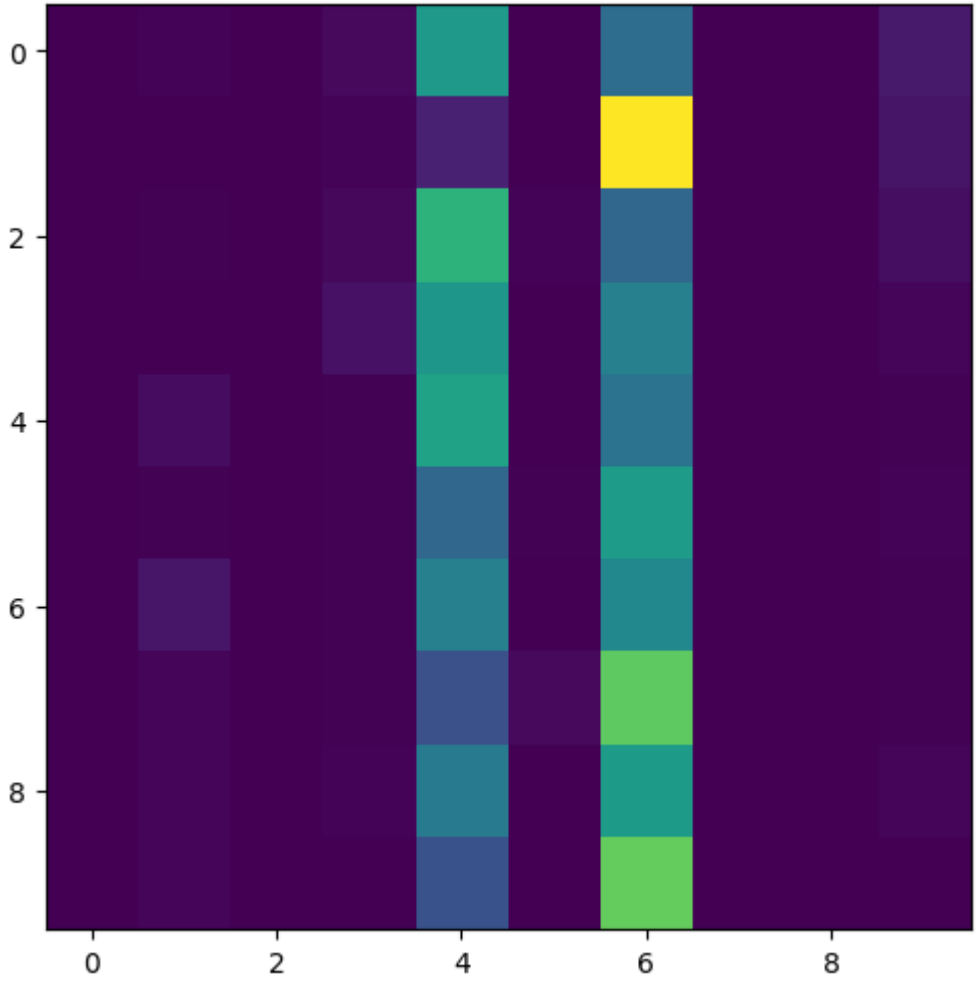
```
        return y_pred


# Initialize the MLP classifier
mlpClassifer = MLPClassifer()

# Train the MLP classifier
# using the training dataset
mlpClassifer.train_net(X_train, y_train, epochs=10, batchSize=50)
```

```
Epoch [1/10], Loss: 0.7019
Epoch [2/10], Loss: 0.9392
Epoch [3/10], Loss: 0.7718
Epoch [4/10], Loss: 0.6448
Epoch [5/10], Loss: 0.5037
Epoch [6/10], Loss: 0.5770
Epoch [7/10], Loss: 0.1281
Epoch [8/10], Loss: 0.3173
Epoch [9/10], Loss: 0.3698
Epoch [10/10], Loss: 0.1709
```

In [29]:
```
# Compute the confusion matrix and
# accuracy for the MLP classifier on
# the test dataset
M_mlp, acc_mlp = Confusion(X_test, y_test, mlpClassifer)

# Print the confusion matrix accuracy
# for the MLP classifier
print('Confusion matrix - MLP classifier accuracy: %f' % acc_mlp)

# Print the overall test accuracy
# of the MLP classifier using the
# test function
print('MLP classifier accuracy: %f' % test(X_test, y_test, mlpClassifer))

# Visualize the confusion
# matrix for the MLP classifier
VisualizeConfussion(M_mlp)

# ================================================================================================
# ATTENTION MR.GIOTIS
```

```
# Παρόλο που το loss είναι χαμηλό, το accuracy παραμένει χαμηλό, κάτι που υποδηλώνει ότι το μοντέλο δεν καταφέρνει
# Αυτό μπορεί να σχετίζεται με προβλήματα στην προεπεξεργασία των δεδομένων, ακατάλληλη αρχικοποίηση των βαρών ή ακ
# την αρχιτεκτονική του μοντέλου, η οποία ενδεχομένως να μην υποστηρίζει αποτελεσματική εκμάθηση.
# ============================================================================================================
```

```
100%|██████████████████████████████████████| 200/200 [00:00<00:00, 2001.06it/s]
Confusion matrix - MLP classifier accuracy: 10.750000
MLP classifier accuracy: 10.750000
```

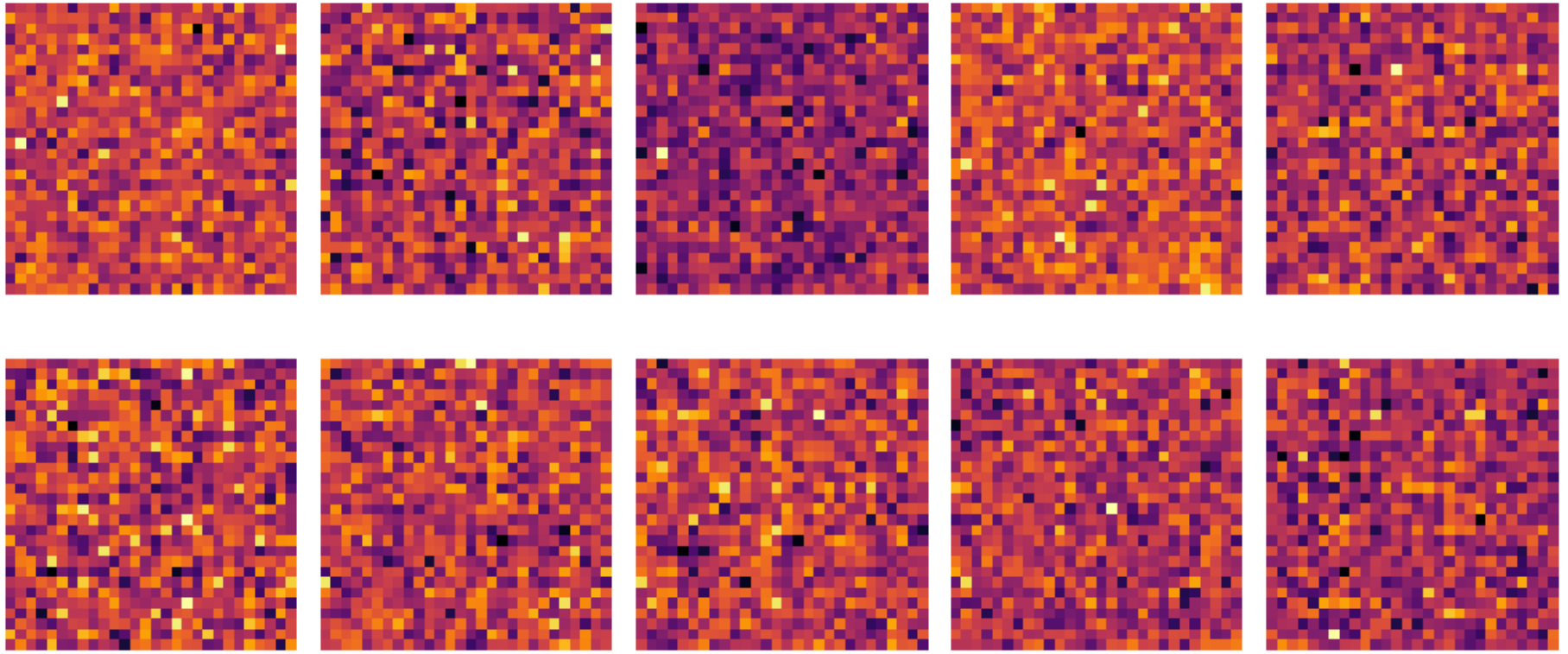


```
[[  0.  10.   0.  24. 528.   0. 349.   0.   0.  69.]
 [  0.   0.   0.   9.  91.   2. 979.   0.   0.  54.]
 [  0.   7.   0.  22. 634.   9. 323.   0.   0.  37.]
 [  0.   0.   0.  47. 519.   3. 426.   0.   0.  15.]
 [  0.  33.   0.   4. 563.   1. 374.   0.   0.   7.]
 [  0.   5.   0.   7. 324.   4. 541.   0.   1.  10.]
 [  0.  59.   0.   6. 427.   3. 458.   0.   0.   5.]
 [  0.  15.   0.   5. 243.  26. 735.   0.   0.   4.]
 [  0.  12.   0.   8. 403.   0. 534.   0.   0.  17.]
 [  0.  13.   0.   1. 245.   1. 746.   0.   0.   3.]]
```

In [30]:

```
# Extract the weights from the first
# layer of the MLP classifier. Ensure
# "MLPweight1" matches the attribute name
# in your class
weights = mlpClassifer.MLPweight1.data.cpu().numpy()

# Create a figure to display the filters
# Adjust the figure size
plt.figure(figsize=(10, 5))

# Iterate through the first 10 outputs
# of the hidden layer
for i in range(10):
    # Reshape the weights corresponding
    # to the i-th output to 28x28 for
    # visualization
    w = weights[i].reshape(28, 28)

    # Normalize the weights to the range
    # [0, 1] for better display contrast
    w_norm = (w - w.min()) / (w.max() - w.min())

    # Determine the position of
    # the filter in the grid
    plt.subplot(2, 5, i + 1)

    # Display the normalized filter
    # using the "inferno" colormap
    plt.imshow(w_norm, cmap='inferno')
```

```
    # Disable axes for a cleaner look
    plt.axis('off')

# Adjust the layout to
# prevent overlapping of subplots
plt.tight_layout()

# Display the figure
# showing the filters
plt.show()
```



Ζήτημα 2.5: Συνελικτικό Νευρωνικό Δίκτυο - Convolutional Neural Network (CNN) [bonus 5 μονάδες]

Εδώ θα υλοποιήσετε ένα CNN με την ακόλουθη αρχιτεκτονική:

- n=10 (output features or filters)
- ReLU(Conv(kernel_size=5x5, stride=2, output_features=n))
- ReLU(Conv(kernel_size=5x5, stride=2, output_features=n*2))
- ReLU(Linear(hidden units = 64))
- Linear(output_features=classes)

Δηλαδή, 2 συνελικτικά επίπεδα (Conv Layers) όπου απεικονίζουν μη-γραμμικά (ReLU) την είσοδο του προηγούμενου επιπέδου, ακολουθούμενα από 1 πλήρως συνδεδεμένο κρυμμένο επίπεδο (FC hidden layer) με μη γραμμική ενεργοποίηση (ReLU) και μετά το επίπεδο εξόδου (output layer) όπου συνδυάζει γραμμικά τις τιμές του προηγούμενου επιπέδου.

Εμφανίστε τον πίνακα σύγχυσης και την ακρίβεια μετά την εκπαίδευση. Θα πρέπει να έχετε περίπου ~98% ακρίβεια για 10 εποχές και μέγεθος παρτίδας 50.

Σημείωση: Δεν επιτρέπεται να χρησιμοποιείτε τις torch.nn.Conv2d() και torch.nn.Linear(). Η χρήση αυτών θα οδηγήσει σε αφαίρεση μονάδων. Χρησιμοποιήστε τις δηλωμένες συναρτήσεις conv2d(), weight_variable() και bias_variable(). Ωστόσο στην πράξη, όταν προχωρήσετε μετά από αυτό το μάθημα, θα χρησιμοποιήσετε torch.nn.Conv2d() που κάνει τη ζωή πιο εύκολη και αποκρύπτει όλες τις υποφαινόμενες λειτουργίες.

Μην ξεχάσετε να σχολιάσετε τον κώδικά σας όπου χρειάζεται (π.χ. στον τρόπο υπολογισμού των διαστάσεων της εξόδου σε κάθε επίπεδο).

In [31]:
```
def conv2d(x, W, stride, bias=None):
    # x: input
    # W: weights (out, in, kH, kW)
    return F.conv2d(x, W, bias, stride=stride, padding=2)

# Defining a Convolutional Neural Network
class CNNClassifer(DNN):
    def __init__(self, classes=10, n=10):
        super(CNNClassifer, self).__init__()
        # Conv1: Conv(kernel_size=5x5, stride=2, output_features=n)
        # n filters, 1 input channel, kernel 5x5
        self.W1 = weight_variable((n, 1, 5, 5))
```

```
        # Bias for n filters
        self.b1 = bias_variable((n,))

        # Conv2: Conv(kernel_size=5x5, stride=2, output_features=n*2)
        # n*2 filters, n input channels, kernel 5x5
        self.W2 = weight_variable((n * 2, n, 5, 5))

        # Bias for n*2 filters
        self.b2 = bias_variable((n * 2,))

        # FC1: Linear(hidden units = 64)
        # Fully connected weights
        self.W_fc1 = weight_variable((64, n * 2 * 7 * 7))

        # Bias for 64 hidden units
        self.b_fc1 = bias_variable((64,))

        # FC2: Linear(output_features=classes)
        # Fully connected output weights
        self.W_fc2 = weight_variable((classes, 64))

        # Bias for output classes
        self.b_fc2 = bias_variable((classes,))

    def forward(self, x):
        # Conv1: Apply convolution, add bias, and ReLU activation
        # Output dims: (batch_size, n, 14, 14)
        x = conv2d(x, self.W1, stride=2, bias=self.b1)
        x = torch.relu(x)

        # Conv2: Apply convolution, add bias, and ReLU activation
        # Output dims: (batch_size, n*2, 7, 7)
        x = conv2d(x, self.W2, stride=2, bias=self.b2)
        x = torch.relu(x)

        # Flatten the output for the fully connected layers
        # Output dims: (batch_size, n*2*7*7)
        x = x.view(x.size(0), -1)

        # FC1: Apply linear transformation and ReLU activation
        # Output dims: (batch_size, 64)
```

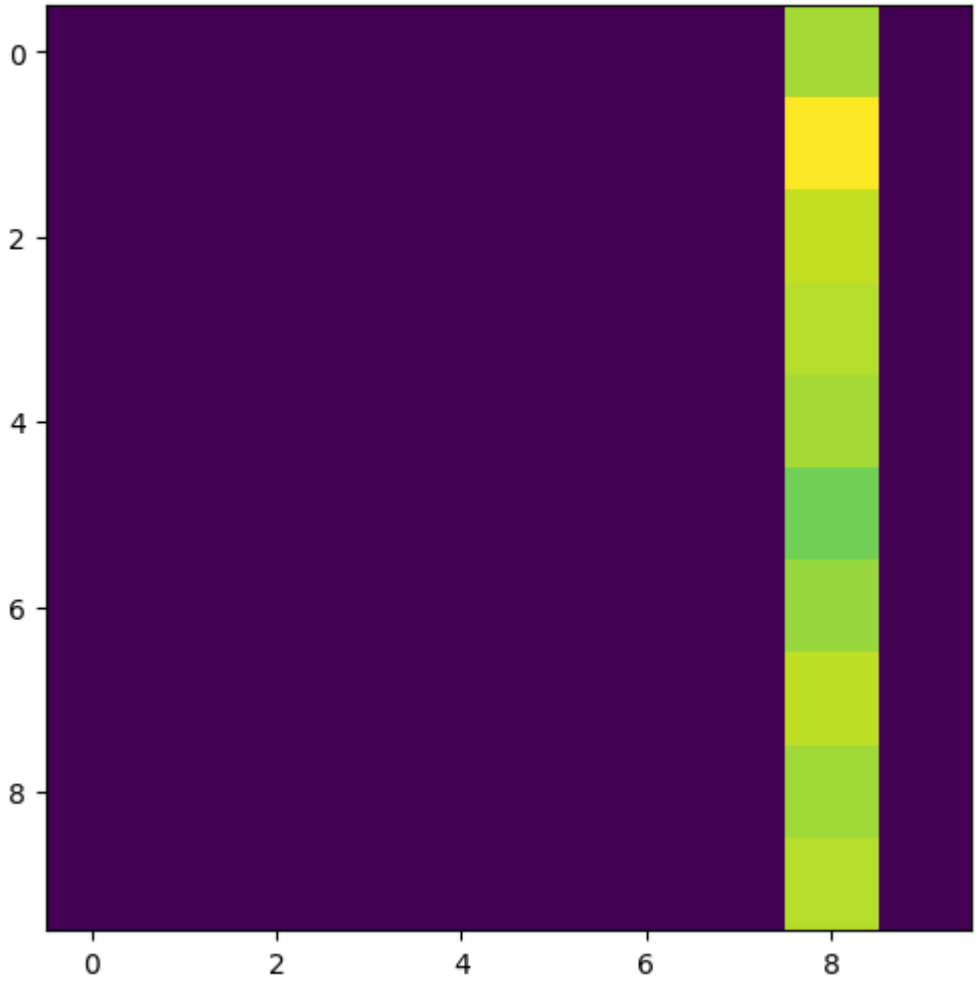
```
        x = torch.addmm(self.b_fc1, x, self.W_fc1.t())
        x = torch.relu(x)

        # FC2: Apply linear transformation for logits
        # Output dims: (batch_size, classes)
        y = torch.addmm(self.b_fc2, x, self.W_fc2.t())

        return y


cnnClassifer = CNNClassifer()
cnnClassifer.train_net(X_train, y_train, epochs=10, batchSize=50)
```

```
Epoch [1/10], Loss: 1.2357
Epoch [2/10], Loss: 0.5522
Epoch [3/10], Loss: 0.6678
Epoch [4/10], Loss: 1.2995
Epoch [5/10], Loss: 0.1313
Epoch [6/10], Loss: 0.4781
Epoch [7/10], Loss: 0.1774
Epoch [8/10], Loss: 0.4263
Epoch [9/10], Loss: 0.2711
Epoch [10/10], Loss: 0.2094
```

In [32]:
```
# Plot confusion matrix and print the
# test accuracy of the classifier. Calculate
# the predictions for the test set
y_pred = cnnClassifer(X_test)

# Compute the confusion matrix
from sklearn.metrics import confusion_matrix
M_cnn = confusion_matrix(y_test, y_pred)

# Compute the accuracy
acc_cnn = np.trace(M_cnn) / np.sum(M_cnn)

print ('Confusion matrix - MLP classifier accuracy: %f'%acc_cnn)

# Check also standard accucary of test() for consistency
print ('MLP classifier accuracy: %f'%test(X_test, y_test, cnnClassifer))
```

```
VisualizeConfussion(M_cnn)
```

```
Confusion matrix - MLP classifier accuracy: 0.097400
MLP classifier accuracy: 9.740000
```



```
[[   0    0    0    0    0    0    0    0  980    0]
 [   0    0    0    0    0    0    0    0 1135    0]
 [   0    0    0    0    0    0    0    0 1032    0]
 [   0    0    0    0    0    0    0    0 1010    0]
 [   0    0    0    0    0    0    0    0  982    0]
 [   0    0    0    0    0    0    0    0  892    0]
 [   0    0    0    0    0    0    0    0  958    0]
 [   0    0    0    0    0    0    0    0 1028    0]
 [   0    0    0    0    0    0    0    0  974    0]
 [   0    0    0    0    0    0    0    0 1009    0]]
```

- Σημειώστε ότι οι προσεγγίσεις MLP/ConvNet οδηγούν σε λίγο μεγαλύτερη ακρίβεια ταξινόμησης από την προσέγγιση K-NN.
- Στη γενική περίπτωση, οι προσεγγίσεις Νευρωνικών Δικτύων οδηγούν σε σημαντική αύξηση της ακρίβειας, αλλά, σε αυτή την περίπτωση, εφόσον το πρόβλημα δεν είναι ιδιαίτερα δύσκολο, η αύξηση της ακρίβειας δεν είναι και τόσο υψηλή.
- Ωστόσο, αυτό εξακολουθεί να είναι αρκετά σημαντικό, δεδομένου του γεγονότος ότι τα ConvNets που χρησιμοποιήσαμε είναι σχετικά απλά, ενώ η ακρίβεια που επιτυγχάνεται χρησιμοποιώντας το K-NN είναι αποτέλεσμα αναζήτησης σε πάνω από 60.000 εικόνες εκπαίδευσης για κάθε εικόνα ελέγχου.
- Συνιστάται ιδιαίτερα να αναζητήσετε περισσότερα για τα νευρωνικά δίκτυα/PyTorch στη διεύθυνση https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html καθώς και στο σχετικό tutorial στην αναρτημένη εργασία στη σελίδα ecourse του μαθήματος **tutorial1_pytorch_introduction.ipynb**.

Οδηγίες υποβολής

Μην ξεχάσετε να κάνετε turnin το αρχείο Jupyter notebook **και** το PDF αρχείο αυτού του notebook μαζί με το συνοδευτικό αρχείο `onoma.txt` : **turnin assignment@mye046 onoma.txt assignment.ipynb assignment.pdf**

Βεβαιωθείτε ότι το περιεχόμενο σε **κάθε κελί εμφανίζεται** καθαρά στο τελικό σας αρχείο PDF. Για να μετατρέψετε το σημειωματάριο σε PDF, μπορείτε να επιλέξετε **έναν** από τους παρακάτω τρόπους:

1. Google Colab (Συνιστάται): You can `print` the web page and save as PDF (e.g. Chrome: Right click the web page → Print... → Choose "Destination: Save as PDF" and click "Save"). Προσοχή στην περίπτωση όπου κώδικας/σχόλια εμφανίζονται εκτός των ορίων της σελίδας. Μια λύση είναι η αλλαγή γραμμής π.χ. σε σχόλια που υπερβαίνουν το πλάτος της σελίδας.

- Στην περίπτωση που οι εικόνες εξόδου δεν εμφανίζονται σωστά, μια λύση μέσω colab είναι (εργαλείο nbconvert):

- Ανέβασμα του αρχείου `assignment.ipynb` στο home directory του Colaboratory (ο κατάλογος home είναι: /content/).
- Εκτελέστε σε ένα κελί colab ενός νέου notebook: `!jupyter nbconvert --to html /content/assignment.ipynb`
- Κάνετε λήψη του assignment.html τοπικά στον υπολογιστή σας και ανοίξτε το αρχείο μέσω browser ώστε να το εξάγετε ως PDF.

2. Local Jupyter/JupyterLab(Συνιστάται): You can `print` the web page and save as PDF (File → Print... → Choose "Destination: Save as PDF" and click "Save"). Προσοχή στην περίπτωση όπου κώδικας/σχόλια εμφανίζονται εκτός των ορίων της σελίδας. Μια λύση είναι η αλλαγή γραμμής π.χ. σε σχόλια που υπερβαίνουν το πλάτος της σελίδας.

3. Local Jupyter/JupyterLab(**Συνιστάται!**): You can `export` and save as HTML (File → Save & Export Notebook as... → HTML). Στη συνέχεια μπορείτε να μετατρέψετε το HTML αρχείο αποθηκεύοντάς το ως PDF μέσω ενός browser.

In [ ]: