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]:
```
import sklearn
sklearn.__version__
```

Out[1]:
```
'1.5.2'
```

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]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
```

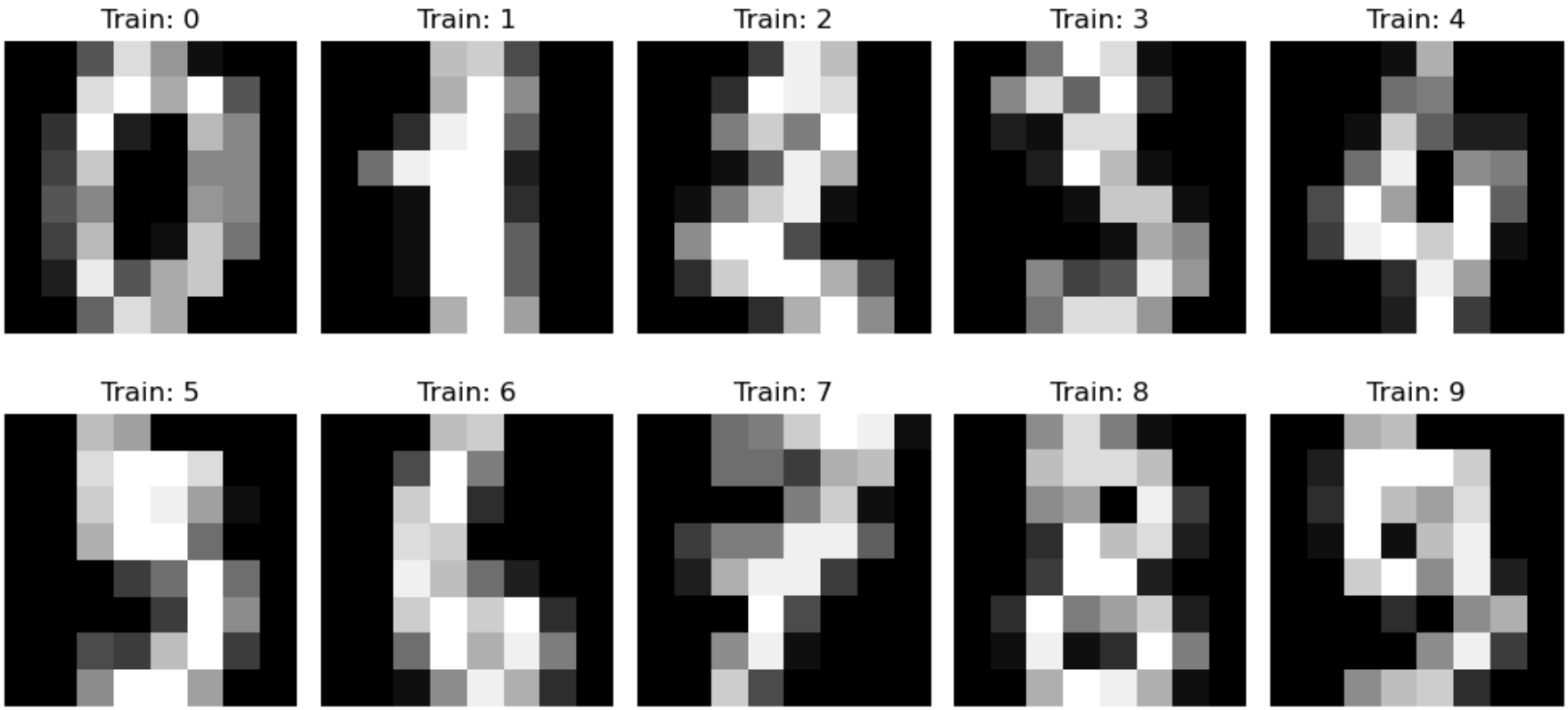
In [3]:
```
# Download MNIST Dataset from Sklearn
digits = datasets.load_digits()

# 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):
    """
    This function plots a sample image for each category,
    in a 2x5 grid.
    """
    # Create a 2x5 grid of subplots
    fig, axes = plt.subplots(2, 5, figsize=(10, 5))

    # Loop through 10 digits and plot them
    for i, ax in enumerate(axes.flat):

        digit_images = digits.images[digits.target == i]  # Get images of the digit

        sample_image = digit_images[0]  # Pick the first image of the digit

        # Display the image in the subplot
        ax.imshow(sample_image, cmap='gray')

        ax.axis('off')  # Hide axis
        ax.set_title(f"Train: {i}")  # Add title

    # Adjust layout and show the plot
    plt.tight_layout()
    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.
    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]:
```
from sklearn.linear_model import LogisticRegression
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):
        self.classes=classes
    def __call__(self, x):
        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
        """
        # Create a LogisticRegression instance from sklearn with the specified solver.
        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,)
        """
        # Fit the logistic regression model to the training data.
        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,)
        """
        # Use the model's predict method to make predictions.
        return self.model.predict(x)
```

```python
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
        """
        # Create a KNeighborsClassifier instance from sklearn with the 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,)
        """
        # Fit the KNN model to the training data.
        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,)
        """
        # Use the model's predict method to make predictions.
        return self.model.predict(x)
```

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

```
Random classifier accuracy: 9.444444
```

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)  # You can adjust k as needed.
knnClassifierX.train(X_train, y_train)

print('k-NN Classifier accuracy: %f' % test(X_test, y_test, knnClassifierX))
```

```
k-NN Classifier accuracy: 96.666667
C:\Users\Pantelis Bonitsis\anaconda3\envs\mye046\lib\site-packages\joblib\externals\loky\backend\context.py:136: UserWarning: Could not find the n
umber of physical cores for the following reason:
found 0 physical cores < 1
Returning the number of logical cores instead. You can silence this warning by setting LOKY_MAX_CPU_COUNT to the number of cores you want to use.
  warnings.warn(
  File "C:\Users\Pantelis Bonitsis\anaconda3\envs\mye046\lib\site-packages\joblib\externals\loky\backend\context.py", line 282, in _count_physical
_cores
    raise ValueError(f"found {cpu_count_physical} physical cores < 1")
```

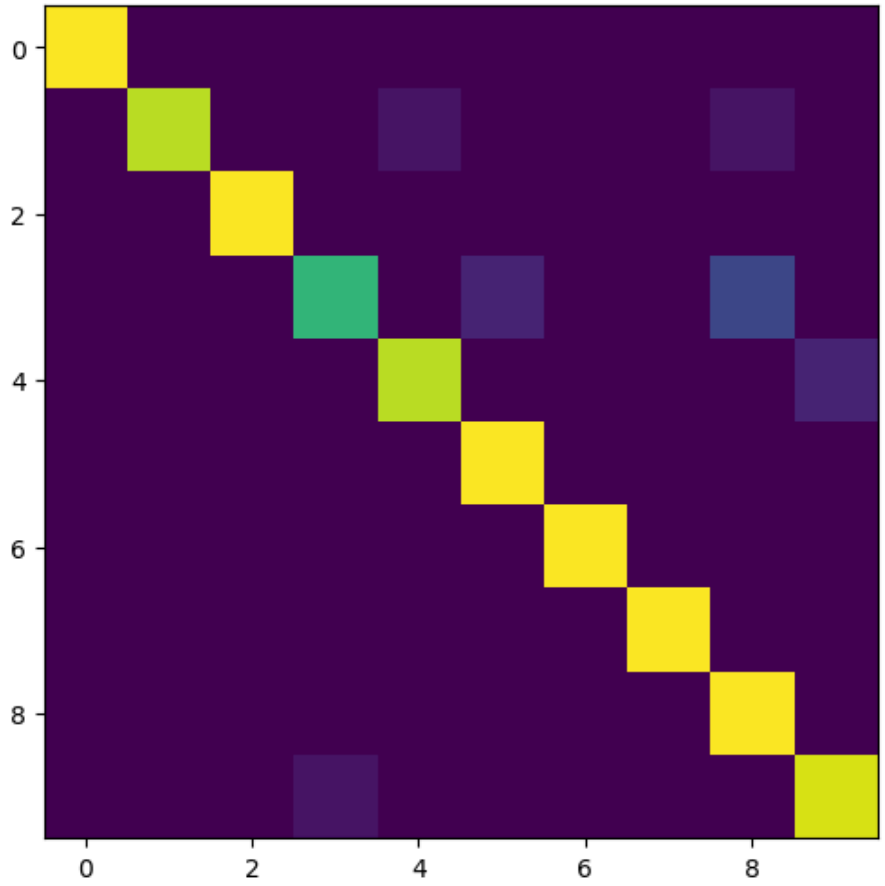
Ζήτημα 1.4: Πίνακας Σύγχυσης [2 μονάδες]

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]:
```
from tqdm import tqdm

def Confusion(testData, testLabels, classifier):
    batchsize=50
    correct=0
    M=np.zeros((10,10))
    num=testData.shape[0]/batchsize
    count=0
    acc=0

    for data, label in tqdm(DataBatch(testData, testLabels, batchsize, shuffle=False), total=len(testData)//batchsize):

        predictions = classifier(data)  # Get predictions from the classifier

        for true_label, pred_label in zip(label, predictions):
            M[true_label, pred_label] += 1  # Increment the matrix at the true vs predicted position

            if true_label == pred_label:  # Check if prediction is correct
                correct += 1  # Increment correct count

            count += 1  # Increment total samples count

    # Normalize each row of the confusion matrix
    for i in range(M.shape[0]):
        row_sum = np.sum(M[i])
        if row_sum > 0:
            M[i] = M[i] / row_sum  # Divide each row element by the row sum

    # Calculate accuracy - Uncomment following line if it is required by your solution above
    acc = correct / count * 100.0

    return M, acc

def VisualizeConfussion(M):
    plt.figure(figsize=(14, 6))
    plt.imshow(M)
    plt.show()
    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, 3981.30it/s]
```

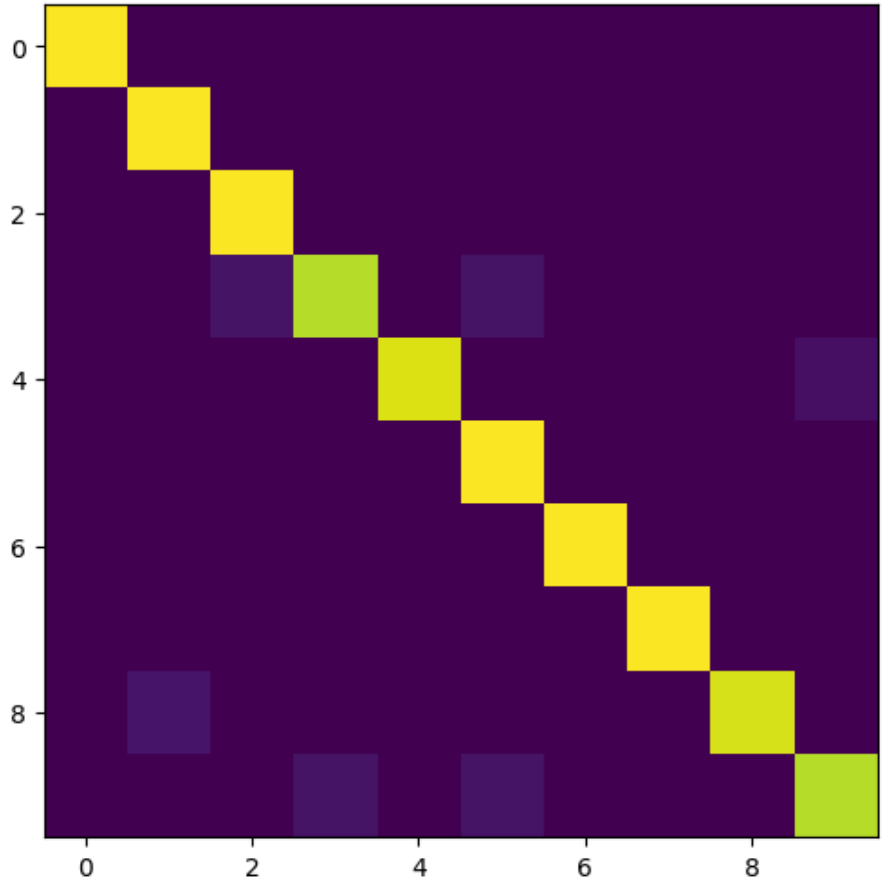



```
[[1.   0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.89 0.   0.   0.05 0.   0.   0.   0.05 0.  ]
 [0.   0.   1.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.67 0.   0.11 0.   0.   0.22 0.  ]
 [0.   0.   0.   0.   0.9  0.   0.   0.   0.   0.1 ]
 [0.   0.   0.   0.   0.   1.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   1.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.   1.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.   0.   1.   0.  ]
 [0.   0.   0.   0.06 0.   0.   0.   0.   0.   0.94]]
```

In [14]:
```
# TEST/PLOT CODE: DO NOT CHANGE
# TEST kNNClassifier
```

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

```
4it [00:00, 210.12it/s]
```




```
[[1.   0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   1.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.   1.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.06 0.89 0.   0.06 0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.95 0.   0.   0.   0.   0.05]
 [0.   0.   0.   0.   0.   1.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   1.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.   1.   0.   0.  ]
 [0.   0.06 0.   0.   0.   0.   0.   0.   0.94 0.  ]
 [0.   0.   0.   0.06 0.   0.06 0.   0.   0.   0.89]]
```

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.

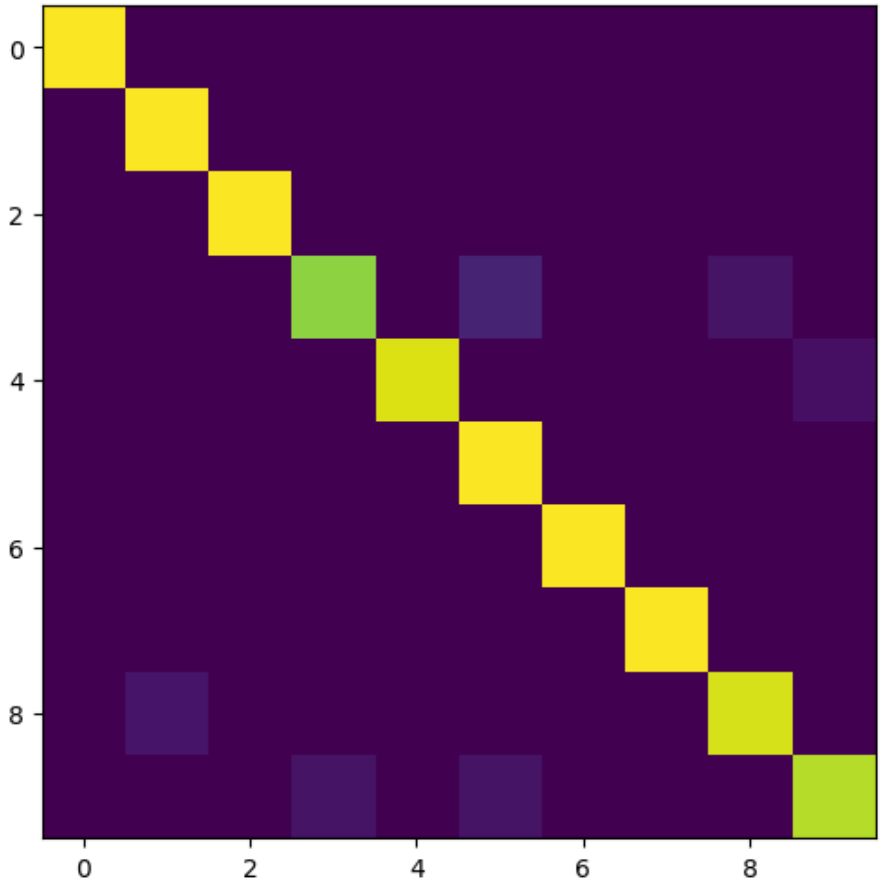
In [15]:
```python
class kNNClassifier_v1_5():
    def __init__(self, k=3):
        self.k = k

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

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

        # Compute the squared differences, sum them, and take the square root
        distance = np.sqrt(np.sum((x1 - x2) ** 2))
        return distance

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

        Returns:
        - indices: Indices of the k nearest neighbors.
        """

        distances = []  # Store distances for all training points

        # Calculate the distance from x to every training point
        for i, train_point in enumerate(self.X_train):
            dist = self.euclidean_distance(x, train_point)
            distances.append((dist, i))  # Save distance and index

         # Sort by distance
        for i in range(len(distances)):
```

```
            for j in range(i + 1, len(distances)):
                if distances[i][0] > distances[j][0]:
                    distances[i], distances[j] = distances[j], distances[i]

        # Select the first k indices
        indices = []
        for i in range(self.k):
            indices.append(distances[i][1])

        return indices


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

        Input:
        - X: Test data array (N, d=64), where N is the number of samples and d is the dimensionality.

        Returns:
        - predicted_labels: Array of predicted labels (N,).
        """
        predicted_labels = []

        # For each nearest k
        # Find the k nearest neighbors
        # Store corresponding Labels of the k neighbors
        # "Vote" for the most frequent label

        for x in X:  # Iterate over each test sample
            # Find the k nearest neighbors for the current sample
            k_indices = self.find_k_nearest_neighbors(x)

            # Get the labels of the k nearest neighbors
            k_labels = []
            for neighbor_index in k_indices:
                label = self.y_train[neighbor_index]
                k_labels.append(label)  # Add the label to the list of labels

            # Count the occurrences of each label
            label_counts = {}
            for label in k_labels:
                if label not in label_counts:
                    label_counts[label] = 0  # Initialize count for new label
                label_counts[label] += 1  # Increment the count
```

```
            # Find the label with the maximum count (majority vote)
            most_common_label = None
            max_count = -1
            for label, count in label_counts.items():
                if count > max_count:
                    most_common_label = label  # Update most common label
                    max_count = count  # Update the maximum count

            # Append the most common label to the predictions
            predicted_labels.append(most_common_label)

        return np.array(predicted_labels)  # Convert the predictions to a NumPy array
```

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))

M, acc = Confusion(X_test, y_test, knnClassifierManualX)

# Display the accuracy and visualize the confusion matrix
print ('kNN classifier accuracy: %f'%test(X_test, y_test, knnClassifierManualX))
VisualizeConfussion(M)
```

```
4it [00:23,  5.81s/it]
kNN classifier accuracy: 96.111111
```




```
[[1.   0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   1.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.   1.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.83 0.   0.11 0.   0.   0.06 0.  ]
 [0.   0.   0.   0.   0.95 0.   0.   0.   0.   0.05]
 [0.   0.   0.   0.   0.   1.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   1.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.   1.   0.   0.  ]
 [0.   0.06 0.   0.   0.   0.   0.   0.   0.94 0.  ]
 [0.   0.   0.   0.06 0.   0.06 0.   0.   0.   0.89]]
```

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):

    # Compute eigenvalues and eigenvectors of A^T * A
    eigenvalues, V = np.linalg.eigh(A.T @ A)

    # Sort eigenvalues and eigenvectors in descending order
    sorted_indices = np.argsort(eigenvalues)[::-1]
    eigenvalues = eigenvalues[sorted_indices]
    V = V[:, sorted_indices]

    # Handle possible zero or negative eigenvalues
    singular_values = np.sqrt(np.maximum(eigenvalues, 0))

    # Compute U using A * V / singular_values
    U = A @ V
    U = np.divide(U, singular_values, where=singular_values > 0)  # Normalize safely

    return U, singular_values, V.T



class PCAKNNClassifer():
    def __init__(self, components=25, k=3):
        """
        Initialize PCA kNN classifier

        Inputs:
```

```
        components: number of principal components
        k: number of neighbors involved in voting
        """
        self.components = components  # Number of principal components
        self.k = k  # Number of neighbors


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

        Inputs:
        trainData: Training images (N,64)
        trainLabels: Labels (N,)
        """

        self.y_train = trainLabels  # Store training labels

        # Center the data by subtracting the mean
        self.mean = np.mean(trainData, axis=0)
        X_hat = trainData - self.mean

        # Perform SVD on centered data
        U, D, V_T = svd(X_hat)

        # Reduce data to the desired number of components
        self.W = V_T[:self.components].T  # Principal components (d x components)
        self.X_train_reduced = X_hat @ self.W  # Project training data onto PCA subspace


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

        Inputs:
        x: Test images (N,64)

        Returns:
        predicted labels (N,)
        """
        # Center the test data using the training mean
        x_centered = x - self.mean

        # Project test data onto PCA subspace
        x_reduced = x_centered @ self.W
```

```
        # Predict labels using kNN in reduced space
        predicted_labels = []
        for test_point in x_reduced:
            # Compute distances to all training points in reduced space
            distances = np.linalg.norm(self.X_train_reduced - test_point, axis=1)

            # Find the indices of the k-nearest neighbors
            k_indices = np.argsort(distances)[:self.k]

            # Get the labels of the k-nearest neighbors
            k_labels = self.y_train[k_indices]

            # Determine the most common label (majority vote)
            label_counts = {}  # Dictionary to count occurrences of each label
            for label in k_labels:
                if label not in label_counts:
                    label_counts[label] = 0  # Initialize count for new label
                label_counts[label] += 1  # Increment the count for the label

            # Find the label with the highest count
            most_common_label = None
            max_count = -1
            for label, count in label_counts.items():
                if count > max_count:
                    most_common_label = label  # Update most common label
                    max_count = count  # Update the maximum count

            predicted_labels.append(most_common_label)

        return np.array(predicted_labels)


# test your classifier with only the first 100 training examples (use this
# while debugging)
pcaknnClassiferX = PCAKNNClassifer()
pcaknnClassiferX.train(X_train[:100], y_train[:100])
print ('PCA-kNN classifier accuracy: %f'%test(X_test, y_test, pcaknnClassiferX))
```

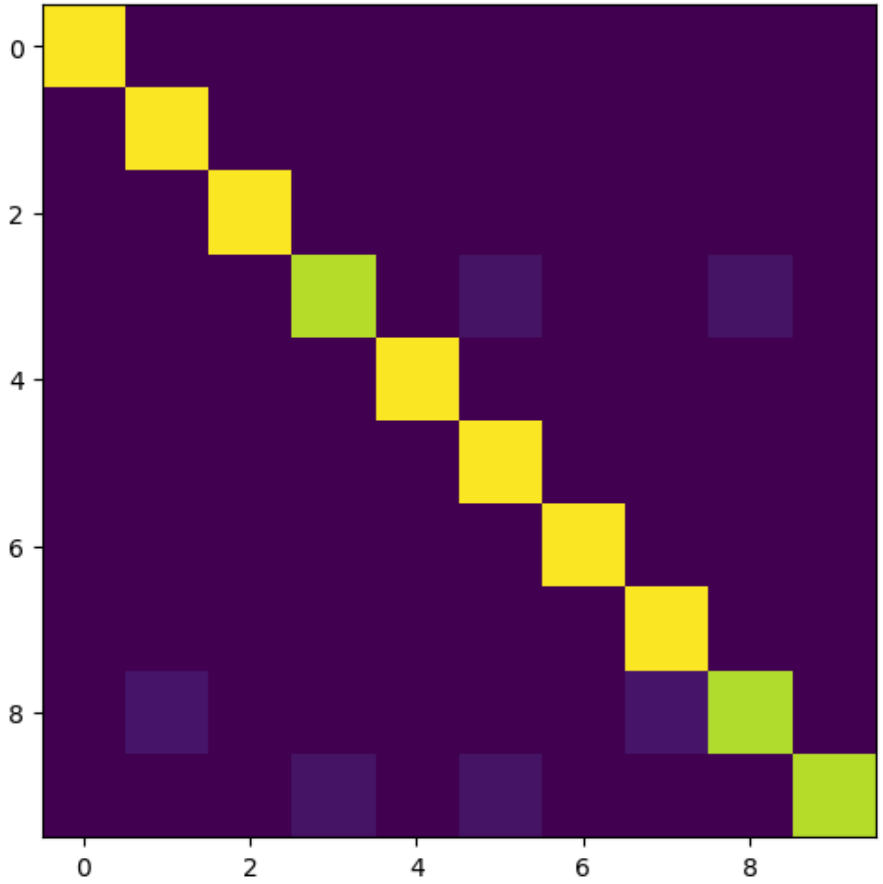
```
PCA-kNN classifier accuracy: 84.444444
```

In [18]:

```
# test your classifier with all the training examples
pcaknnClassifer = PCAKNNClassifer()
pcaknnClassifer.train(X_train, y_train)

# display confusion matrix for your PCA KNN classifier with all the training examples
```

```
M_pca, acc_pca = Confusion(X_test, y_test, pcaknnClassifer)

# Display the accuracy and visualize the confusion matrix
print ('PCA-kNN classifier accuracy: %f'%test(X_test, y_test, pcaknnClassifer))
VisualizeConfussion(M_pca)
```

```
4it [00:00, 141.56it/s]
PCA-kNN classifier accuracy: 96.666667
```



```
[[1.   0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   1.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.   1.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.89 0.   0.06 0.   0.   0.06 0.  ]
 [0.   0.   0.   0.   1.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   1.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   1.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.   1.   0.   0.  ]
 [0.   0.06 0.   0.   0.   0.   0.   0.06 0.88 0.  ]
 [0.   0.   0.   0.06 0.   0.06 0.   0.   0.   0.89]]
```

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

Ο ταξινομητής PCA-kNN (4it [00:00, 141.56it/s]) είναι αρκετά ταχύτερος από τον χειροποίητο kNN (1.5) (4it [00:23, 5.81s/it = 0.1721it/s]). Το PCA μειώνει τις διαστάσεις, κάνοντας την εύρεση γειτόνων πιο αποδοτική. Αν και ο χειροποίητος kNN είναι πιο αργός λόγω απλοϊκής υλοποίησης, το μικρό μέγεθος των δεδομένων περιορίζει την επίδραση της μείωσης διαστάσεων, για μεγαλύτερα σύνολα δεδομένων το PCA θα είχε ακόμα μεγαλύτερο όφελος στον χρόνο εκτέλεσης.

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]:
```
import scipy
import torch.nn as nn
import torch.nn.functional as F
import torch
from torch.autograd import Variable

# create a tensor of 5x3 random-valued array
x = torch.rand(5, 3)
print(x)
```

```
tensor([[0.8895, 0.7738, 0.5902],
        [0.5423, 0.5680, 0.8202],
        [0.4241, 0.4508, 0.4690],
        [0.7210, 0.4360, 0.6045],
        [0.1384, 0.3481, 0.6218]])
```

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/'**.

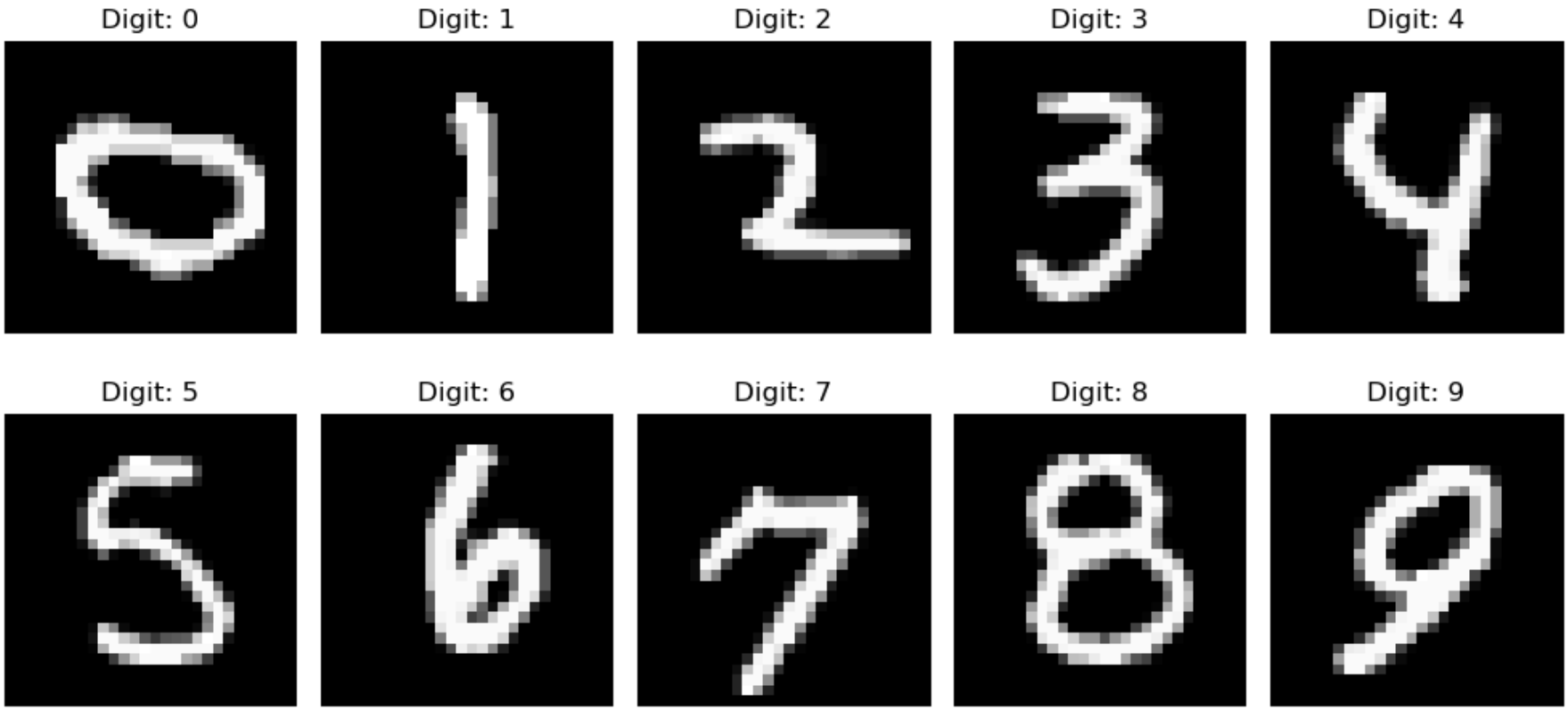
- Θα πρέπει να απεικονίσετε σε ένα σχήμα 2x5 ένα τυχαίο παράδειγμα εικόνας που αντιστοιχεί σε κάθε ετικέτα (κατηγορία) από τα δεδομένα εκπαίδευσης (αντίστοιχα του ζητήματος 1.2).

In [20]:
```python
import torch
import torchvision.datasets as datasets

# import additional libs in case not already done in 'askisi 1'
import matplotlib.pyplot as plt
import numpy as np

# Define the dataset directory
path = 'C:/Users/Pantelis Bonitsis/Desktop/Uoi_Cse/orasi/ex/assignment/mnist/'

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

# Extract the images and labels from the training dataset
X_train = train_dataset.data.numpy()
y_train = train_dataset.targets.numpy()

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

# Extract the images and labels from the testing dataset
X_test = test_dataset.data.numpy()
y_test = test_dataset.targets.numpy()
```

In [21]:
```python
def plot_mnist_sample_high_res(X_train, y_train):
    """
    This function plots a sample image for each category,
    The result is a figure with 2x5 grid of images.

    """
    # Create a 2x5 grid of subplots
    fig, axes = plt.subplots(2, 5, figsize=(10, 5))

    # Loop through the 10 categories (digits 0-9)
    for digit, ax in enumerate(axes.flat):
        # Find indices of all images for the current digit
        digit_indices = np.where(y_train == digit)[0]

        # Randomly select one index from the available indices
        random_index = np.random.choice(digit_indices)

        # Get the corresponding image
        random_image = X_train[random_index]
```

```
        # Display the image in the subplot
        ax.imshow(random_image, cmap='gray')  # Show the grayscale image

        ax.axis('off')  # Hide axis
        ax.set_title(f"Digit: {digit}")  # Add title

    # Adjust layout and show the plot
    plt.tight_layout()
    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]:
```
# base class for your deep neural networks. It implements the training loop (train_net).


import torch.nn.init
import torch.optim as optim
from torch.autograd import Variable
from torch.nn.parameter import Parameter
from tqdm import tqdm
from scipy.stats import truncnorm

class DNN(nn.Module):
    def __init__(self):
        super(DNN, self).__init__()
        pass

    def forward(self, x):
        raise NotImplementedError

    def train_net(self, X_train, y_train, epochs=1, batchSize=50):

        # criterion selection, i.e, loss function
        criterion = torch.nn.CrossEntropyLoss()
```

```
        # optimizer selection, using `optim.`
        optimizer = optim.SGD(self.parameters(), lr=0.01)
        #optimizer = optim.Adam(self.parameters(), lr=0.01)

         # For each epoch
        for epoch in range(epochs):

            # For each batch
            for i in range(0, len(X_train), batchSize):

                # Assign inputs and labels using PyTorch's autograd package via Variable
                inputs = Variable(torch.FloatTensor(X_train[i:i+batchSize]))
                labels = Variable(torch.LongTensor(y_train[i:i+batchSize]))

                # Forward pass
                outputs = self.forward(inputs)

                # Compute loss
                loss = criterion(outputs, labels)

                # Backward pass
                optimizer.zero_grad()  # Clear gradients
                loss.backward()  # Compute gradients
                optimizer.step()  # Update weights

            # Print loss at the end of each epoch
            print(f"Epoch {epoch + 1}/{epochs}, Loss: {loss.item()}")


    def __call__(self, x):
        inputs = Variable(torch.FloatTensor(x))
        prediction = self.forward(inputs)
        return np.argmax(prediction.data.cpu().numpy(), 1)

# helper function to get weight variable
def weight_variable(shape):
    initial = torch.Tensor(truncnorm.rvs(-1/0.01, 1/0.01, scale=0.01, size=shape))
    return Parameter(initial, requires_grad=True)

# helper function to get bias variable
def bias_variable(shape):
    initial = torch.Tensor(np.ones(shape)*0.1)
    return Parameter(initial, requires_grad=True)
```

In [24]:
```
# example linear classifier - input connected to output
# you can take this as an example to learn how to extend DNN class
class LinearClassifier(DNN):
    def __init__(self, in_features=28*28, classes=10):
        super(LinearClassifier, self).__init__()
        # in_features=28*28
        self.weight1 = weight_variable((classes, in_features))
        self.bias1 = bias_variable((classes))

    def forward(self, x):
        # linear operation
        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)
```

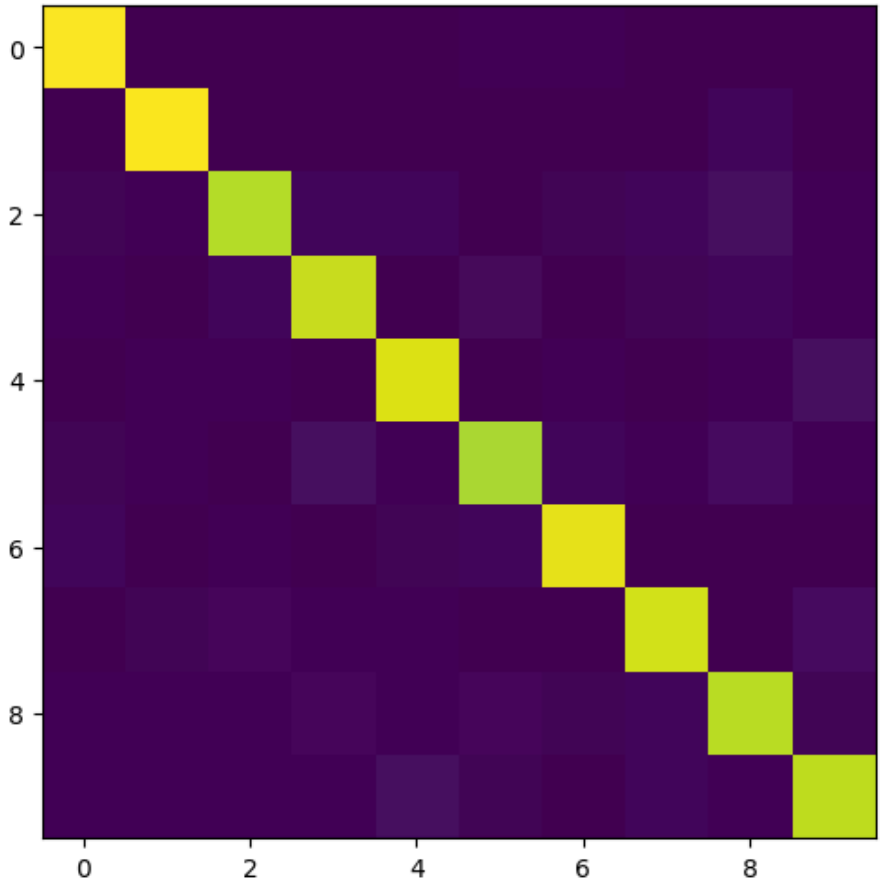
In [25]:
```
# test the example linear classifier (note you should get around 90% accuracy
# for 10 epochs and batchsize 50)
linearClassifier = LinearClassifier()
linearClassifier.train_net(X_train, y_train, epochs=10)

print ('Linear classifier accuracy: %f'%test(X_test, y_test, linearClassifier))
```

```
Epoch 1/10, Loss: 0.5037558078765869
Epoch 2/10, Loss: 0.34552204608917236
Epoch 3/10, Loss: 0.278782457113266
Epoch 4/10, Loss: 0.24143661558628082
Epoch 5/10, Loss: 0.2175101488828659
Epoch 6/10, Loss: 0.20084474980831146
Epoch 7/10, Loss: 0.18854136765003204
Epoch 8/10, Loss: 0.17905668914318085
Epoch 9/10, Loss: 0.17149530351161957
Epoch 10/10, Loss: 0.16530366241931915
Linear classifier accuracy: 91.210000
```

In [26]:

```python
# display confusion matrix
M, acc = Confusion(X_test, y_test, linearClassifier)
print('Linear classifier accuracy: %f' % acc)
VisualizeConfussion(M)
```

```
100%|██████████| 200/200 [00:00<00:00, 7265.88it/s]
Linear classifier accuracy: 91.210000
```



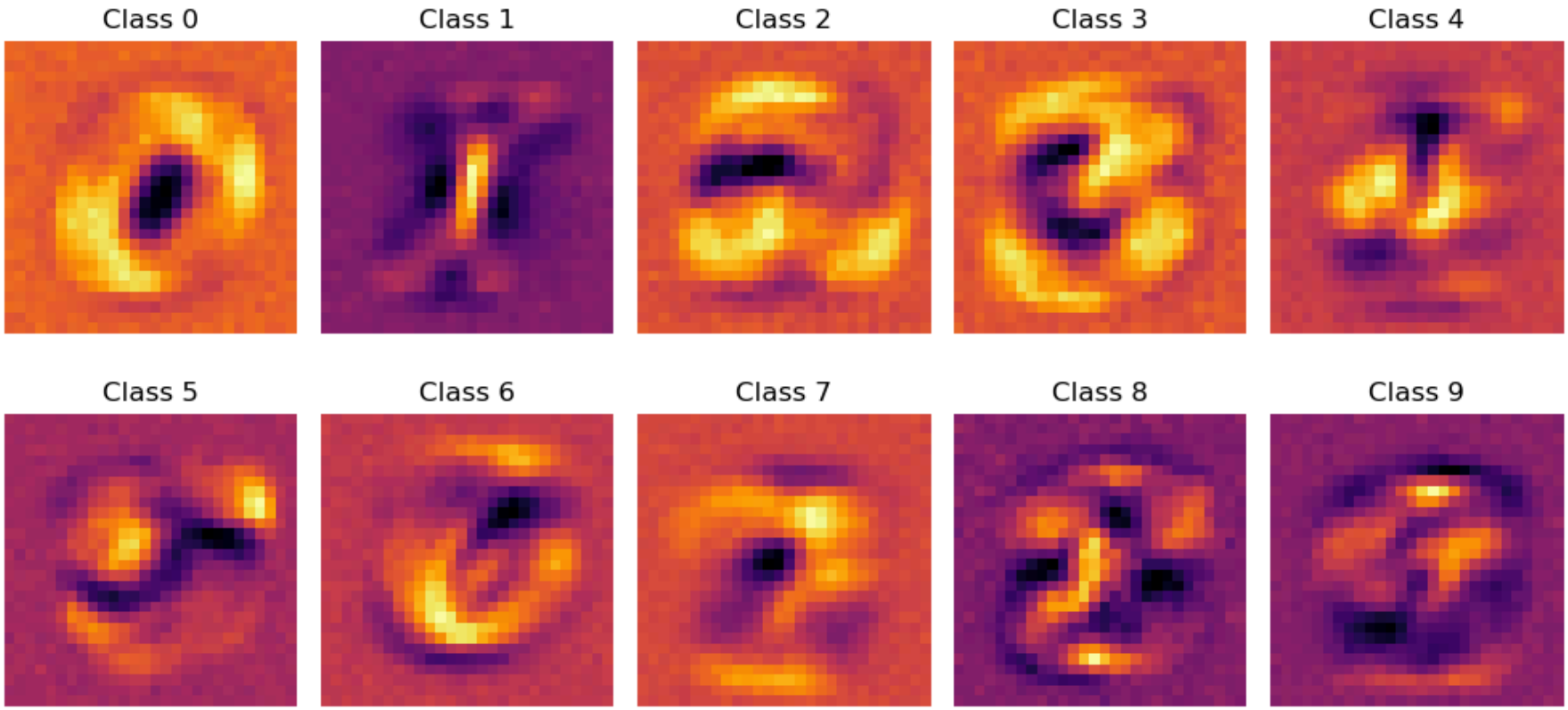

```
[[0.98 0.   0.   0.   0.   0.   0.01 0.   0.   0.  ]
 [0.   0.97 0.   0.   0.   0.   0.   0.   0.02 0.  ]
 [0.01 0.01 0.87 0.02 0.02 0.   0.01 0.02 0.04 0.01]
 [0.   0.   0.02 0.9  0.   0.03 0.   0.01 0.02 0.01]
 [0.   0.   0.01 0.   0.93 0.   0.01 0.   0.01 0.04]
 [0.01 0.   0.   0.04 0.01 0.85 0.02 0.01 0.03 0.01]
 [0.02 0.   0.   0.   0.01 0.02 0.94 0.   0.   0.  ]
 [0.   0.01 0.02 0.01 0.01 0.   0.   0.91 0.   0.03]
 [0.01 0.01 0.01 0.03 0.01 0.02 0.01 0.02 0.88 0.01]
 [0.01 0.01 0.   0.01 0.04 0.01 0.   0.02 0.01 0.89]]
```

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]:
```
# Plot filter weights corresponding to each class, you may have to reshape them to make sense out of them
# linearClassifier.weight1.data will give you the first layer weights

# Access the weights from the first layer of the classifier
weights = linearClassifier.weight1.data.numpy()  # Convert to numpy for easier manipulation

# Normalize weights for visualization
weights_min = weights.min(axis=1, keepdims=True)  # Minimum value of each row (class)
weights_max = weights.max(axis=1, keepdims=True)  # Maximum value of each row (class)
weights_shifted = weights - weights_min  # Shift weights to start from 0
normalized_weights = weights_shifted / (weights_max - weights_min)

# Plot the weights as images
fig, axes = plt.subplots(2, 5, figsize=(10, 5))  # 2x5 grid for 10 classes

for i, ax in enumerate(axes.flat):
    # Reshape each class weight to 28x28
    weight_image = normalized_weights[i].reshape(28, 28)

    # Display the weight image
    im = ax.imshow(weight_image, cmap='inferno')  # Use 'inferno' colormap
    ax.axis('off')  # Hide axes
    ax.set_title(f"Class {i}")  # Title for each class

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```



Σχολιασμός των βαρών

Τα βάρη μοιάζουν με θολές αναπαραστάσεις των ψηφίων επειδή ο ταξινομητής προσπαθεί να εντοπίσει τα χαρακτηριστικά που είναι πιο συχνά σε κάθε κατηγορία. Οι τιμές των βαρών αντιπροσωπεύουν τα μοτίβα που συμβάλλουν περισσότερο στη σωστή ταξινόμηση κάθε ψηφίου.

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):
    def __init__(self, in_features=28*28, classes=10, hidden=50):
        """
        Initialize weight and bias variables
        """
        super(MLPClassifer, self).__init__()

        # Initialize weights and biases for input to hidden layer
        self.weight1 = weight_variable((hidden, in_features))  # Hidden layer weights
        self.bias1 = bias_variable((hidden,))  # Hidden layer bias

        # Initialize weights and biases for hidden to output layer
        self.weight2 = weight_variable((classes, hidden))  # Output layer weights
        self.bias2 = bias_variable((classes,))  # Output layer bias


    def forward(self, x):

        # Flatten the input
        x = torch.flatten(x,1)

        # Compute hidden layer activations
        hidden_output = torch.matmul(x, self.weight1.T) + self.bias1  # Linear transformation
        hidden_output = torch.relu(hidden_output)  # ReLU activation
```

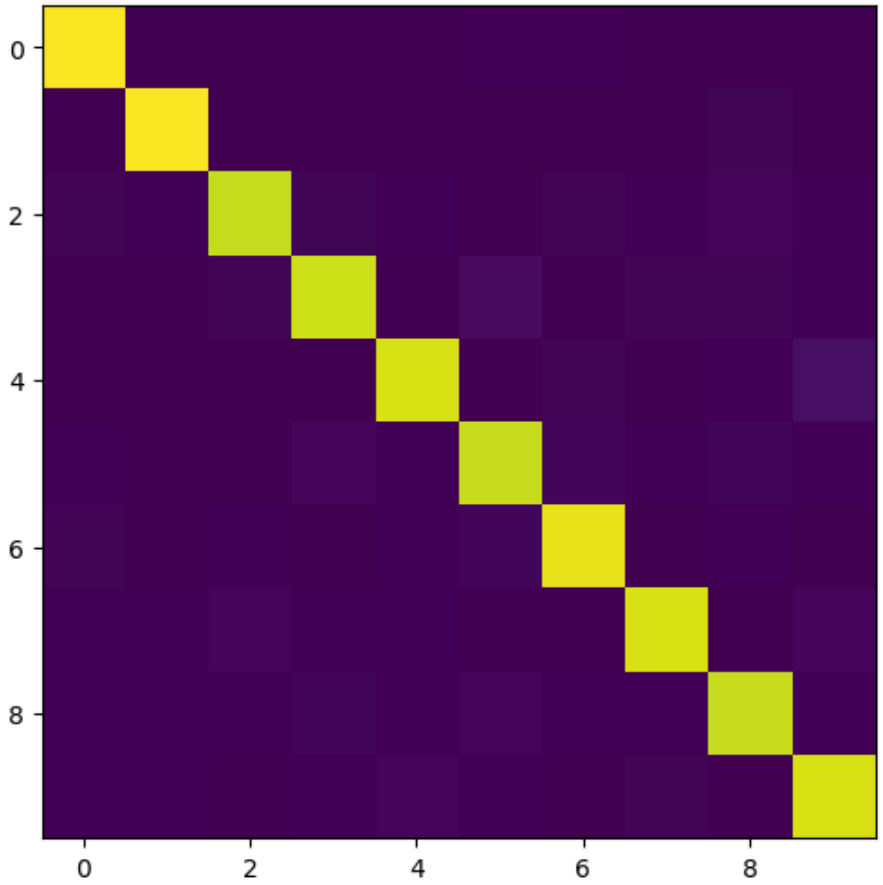
```
        # Compute output layer activations (logits)
        logits = torch.matmul(hidden_output, self.weight2.T) + self.bias2  # Linear transformation
        return logits



mlpClassifer = MLPClassifer()
mlpClassifer.train_net(X_train, y_train, epochs=10, batchSize=50)
```

```
Epoch 1/10, Loss: 0.7959820628166199
Epoch 2/10, Loss: 0.3482176661491394
Epoch 3/10, Loss: 0.24293021857738495
Epoch 4/10, Loss: 0.19453200697898865
Epoch 5/10, Loss: 0.16740097105503082
Epoch 6/10, Loss: 0.14950430393218994
Epoch 7/10, Loss: 0.13634414970874786
Epoch 8/10, Loss: 0.12117113918066025
Epoch 9/10, Loss: 0.10674399137496948
Epoch 10/10, Loss: 0.09480486810207367
```

In [29]:

```
# Plot confusion matrix
M_mlp,acc_mlp = Confusion(X_test, y_test, mlpClassifer)

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

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

VisualizeConfussion(M_mlp)
```

```
100%|██████████| 200/200 [00:00<00:00, 4993.52it/s]
Confusion matrix - MLP classifier accuracy: 92.950000
MLP classifier accuracy: 92.950000
```

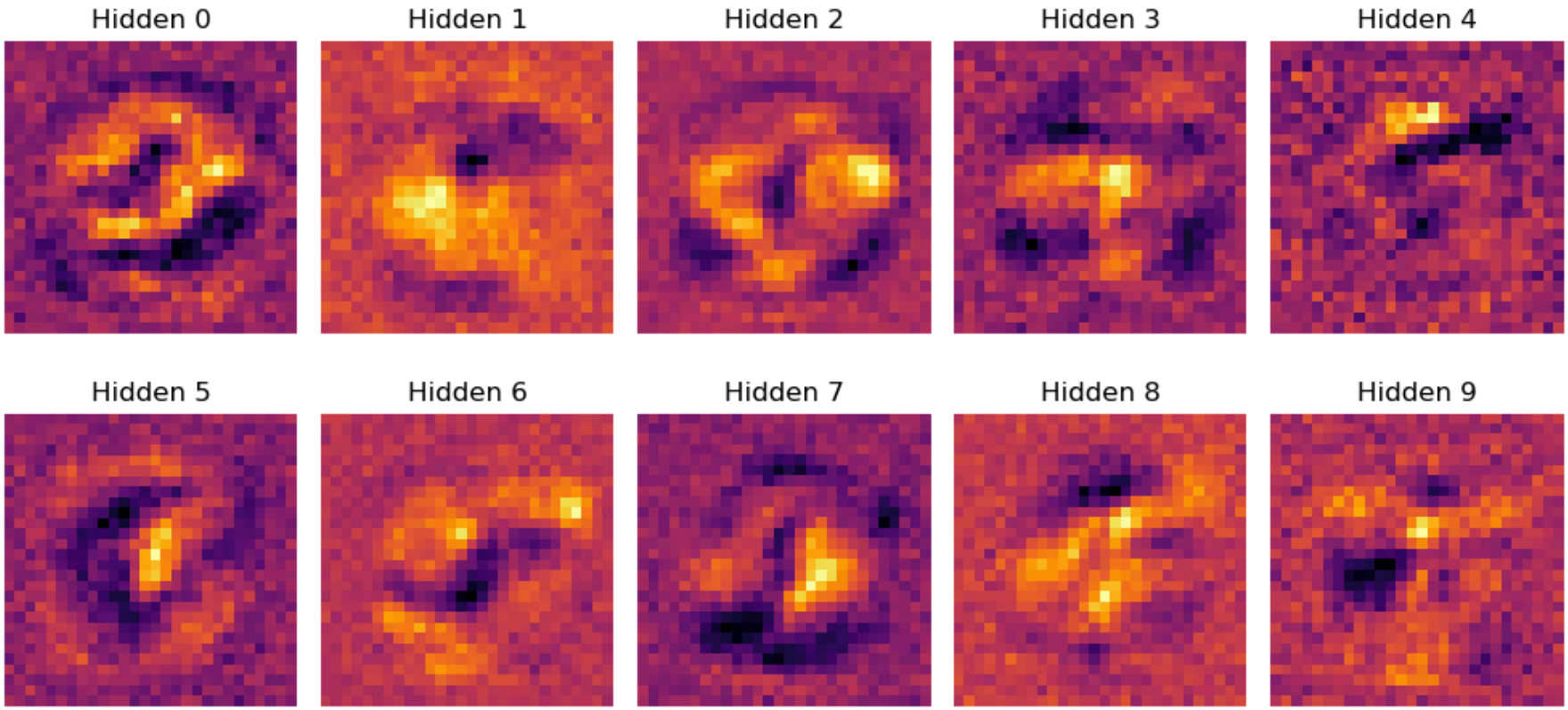



```
[[0.98 0.   0.   0.   0.   0.01 0.01 0.   0.   0.  ]
 [0.   0.98 0.   0.   0.   0.   0.   0.   0.01 0.  ]
 [0.01 0.01 0.9  0.01 0.01 0.   0.01 0.01 0.03 0.01]
 [0.   0.   0.01 0.91 0.   0.03 0.   0.01 0.01 0.01]
 [0.   0.   0.   0.   0.92 0.   0.01 0.   0.01 0.05]
 [0.01 0.   0.   0.02 0.01 0.9  0.02 0.01 0.02 0.01]
 [0.01 0.   0.   0.   0.01 0.02 0.95 0.   0.   0.  ]
 [0.   0.01 0.02 0.01 0.   0.   0.   0.92 0.   0.03]
 [0.01 0.01 0.01 0.02 0.01 0.03 0.01 0.01 0.9  0.01]
 [0.01 0.01 0.   0.01 0.03 0.01 0.   0.01 0.   0.92]]
```

In [30]:

```
# Plot filter weights

# Access the weights from the hidden layer of the MLP classifier
weights_hidden = mlpClassifer.weight1.data.numpy()  # Convert to numpy for easier manipulation
```

```
# Normalize weights for visualization
weights_min = weights_hidden.min(axis=1, keepdims=True)  # Minimum value of each row (hidden unit)
weights_max = weights_hidden.max(axis=1, keepdims=True)  # Maximum value of each row (hidden unit)
weights_shifted = weights_hidden - weights_min  # Shift weights to start from 0
normalized_weights = weights_shifted / (weights_max - weights_min)  # Normalize to [0, 1]

# Plot the first 10 filters from the hidden layer
fig, axes = plt.subplots(2, 5, figsize=(10, 5))  # 2x5 grid for the first 10 filters

for i, ax in enumerate(axes.flat):
    # Reshape each filter weight to 28x28
    weight_image = normalized_weights[i].reshape(28, 28)

    # Display the weight image
    im = ax.imshow(weight_image, cmap='inferno')  # Use 'inferno' colormap
    ax.axis('off')  # Hide axes
    ax.set_title(f"Hidden {i}")  # Title for each filter

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```



Σχολιασμός των βαρών

Τα βάρη του MLP δεν μοιάζουν με αυτά του γραμμικού ταξινομητή, καθώς αποτυπώνουν πιο αφηρημένα χαρακτηριστικά από τα δεδομένα εισόδου.Αυτό συμβαίνει επειδή τα βάρη του MLP αποτυπώνουν πιο σύνθετες και αφηρημένες αναπαραστάσεις στο κρυμμένο επίπεδο, αντί για άμεσες αναπαραστάσεις των ψηφίων.

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).

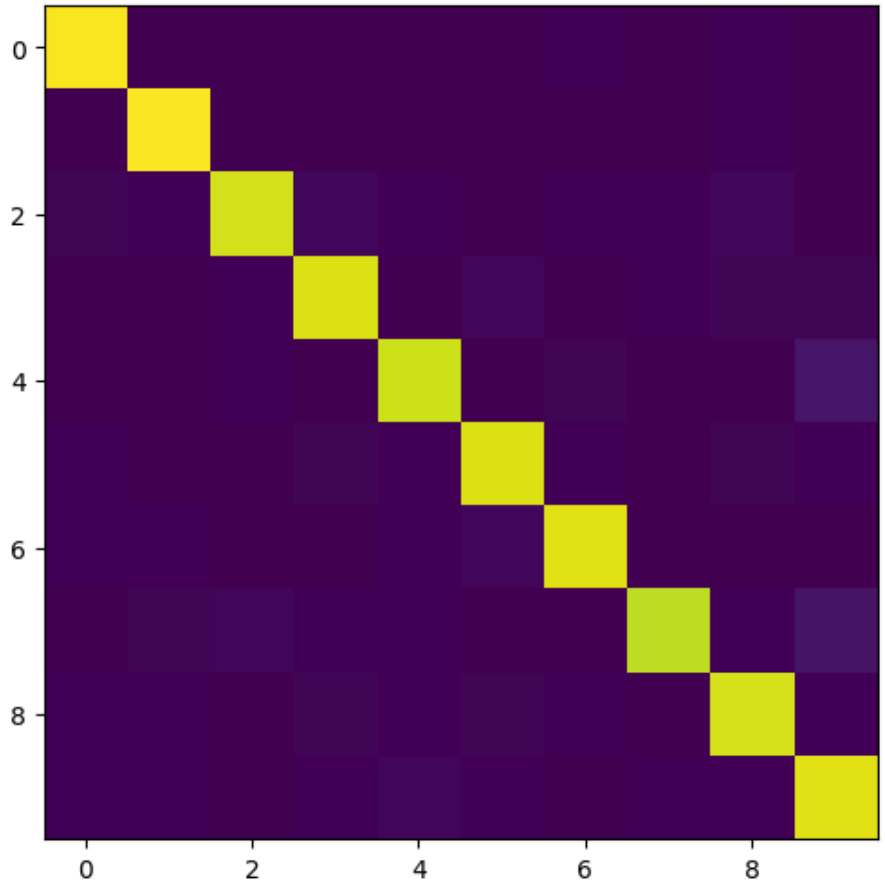
In [33]:
```
def conv2d(x, W, stride, bias=None):
    # x: input
    # W: weights (out, in, kH, kW)
    return F.conv2d(x, W, bias, stride=stride, padding=2)

# Defining a Convolutional Neural Network
class CNNClassifer(DNN):
    def __init__(self, classes=10, n=10):
        super(CNNClassifer, self).__init__()

        # Initialize weights and biases for the first convolutional layer
        self.weight1 = weight_variable((n, 1, 5, 5))  # n filters, input_channels=1, kernel=5x5
        self.bias1 = bias_variable((n,))

        # Initialize weights and biases for the second convolutional layer
        self.weight2 = weight_variable((n * 2, n, 5, 5))  # n*2 filters, input_channels=n, kernel=5x5
        self.bias2 = bias_variable((n * 2,))

        # Initialize weights and biases for the fully connected hidden layer
        self.weight3 = weight_variable((64, 7 * 7 * n * 2))
        self.bias3 = bias_variable((64,))

        # Initialize weights and biases for the output layer
        self.weight4 = weight_variable((classes, 64))  # Output layer weights
        self.bias4 = bias_variable((classes,))

    def forward(self, x):
```

```
        # First convolutional layer
        x = conv2d(x, self.weight1, stride=2, bias=self.bias1)  # Apply convolution
        x = F.relu(x)  # Apply ReLU activation

        # Second convolutional layer
        x = conv2d(x, self.weight2, stride=2, bias=self.bias2)  # Apply convolution
        x = F.relu(x)  # Apply ReLU activation

        # Flatten the output for the fully connected layer
        x = torch.flatten(x,1)

        # Fully connected hidden layer
        x = torch.matmul(x, self.weight3.T) + self.bias3  # Apply linear transformation
        x = F.relu(x)  # Apply ReLU activation

        # Output layer
        y = torch.matmul(x, self.weight4.T) + self.bias4  # Apply linear transformation for final output

        return y

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

```
Epoch 1/10, Loss: 2.3007259368896484
Epoch 2/10, Loss: 2.300755262374878
Epoch 3/10, Loss: 2.300713062286377
Epoch 4/10, Loss: 2.3005754947662354
Epoch 5/10, Loss: 2.3000710010528564
Epoch 6/10, Loss: 2.296143054962158
Epoch 7/10, Loss: 0.49337416887283325
Epoch 8/10, Loss: 0.1389007270336151
Epoch 9/10, Loss: 0.052835747599601746
Epoch 10/10, Loss: 0.03081592358648777
```

In [34]:
```
# Plot confusion matrix and print the test accuracy of the classifier

M_cnn, acc_cnn = Confusion(X_test, y_test, cnnClassifer)

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)
```

```
100%|██████████| 200/200 [00:00<00:00, 1673.49it/s]
Confusion matrix - MLP classifier accuracy: 94.260000
MLP classifier accuracy: 94.260000
```




```
[[0.99 0.   0.   0.   0.   0.   0.01 0.   0.   0.  ]
 [0.   0.99 0.   0.   0.   0.   0.   0.   0.01 0.  ]
 [0.01 0.01 0.92 0.02 0.   0.   0.01 0.   0.02 0.  ]
 [0.   0.   0.01 0.94 0.   0.02 0.   0.01 0.01 0.01]
 [0.   0.   0.   0.   0.92 0.   0.01 0.   0.   0.06]
 [0.01 0.   0.   0.01 0.01 0.94 0.01 0.   0.01 0.01]
 [0.01 0.   0.   0.   0.01 0.02 0.95 0.   0.   0.  ]
 [0.   0.01 0.02 0.01 0.   0.   0.   0.9  0.   0.06]
 [0.01 0.01 0.   0.01 0.01 0.01 0.01 0.   0.93 0.01]
 [0.   0.01 0.   0.   0.02 0.01 0.   0.   0.01 0.95]]
```

- 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.