

# 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 βιβλιοθήκης 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 αποσυμπίεσετε 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
```

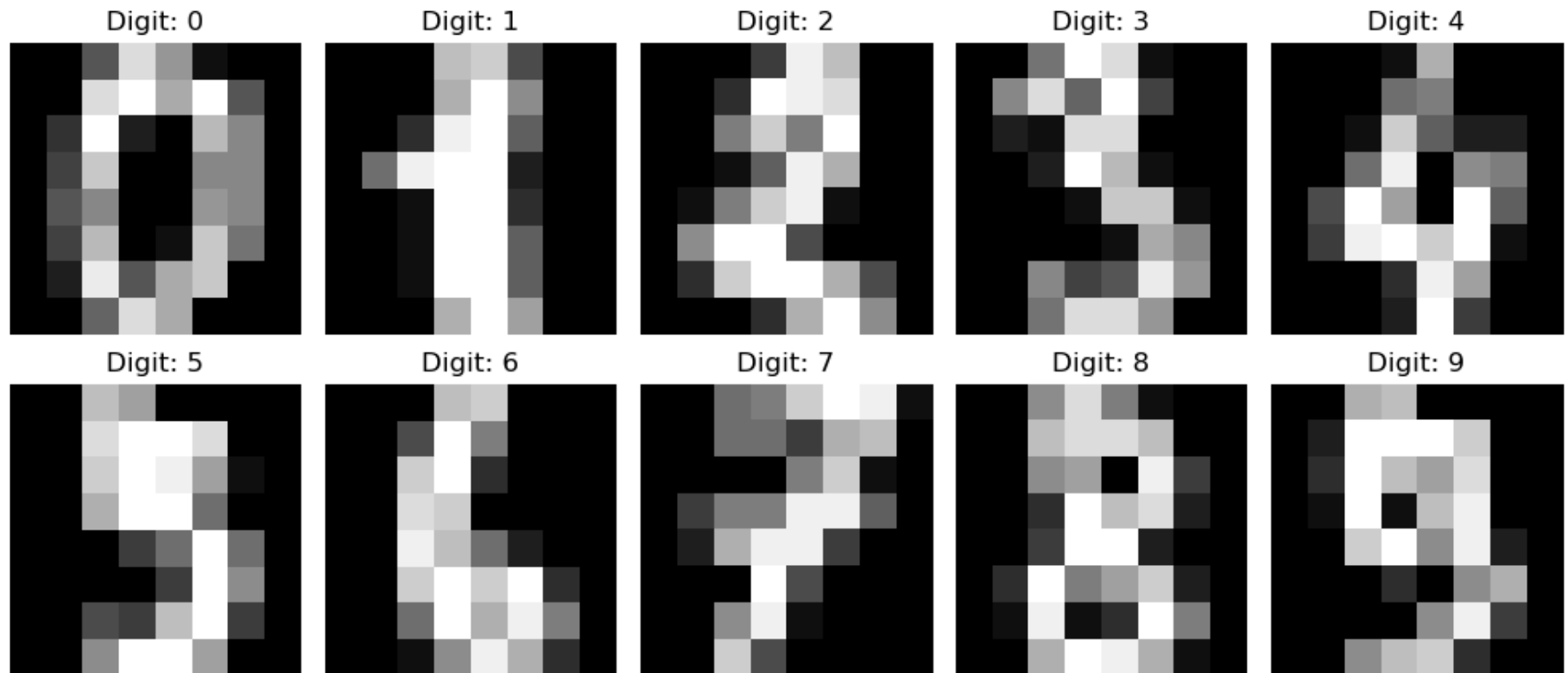
```
In [3]: # Download MNIST Dataset from Sklearn  
digits = datasets.load_digits()  
  
# Print to show there are 1797 images (8 by 8)  
print("Images Shape" , digits.images.shape)  
  
# Print to show there are 1797 image data (8 by 8 images for a dimensionality of 64)  
print("Image Data Shape" , 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.  
        The result is a figure with a 2x5 grid of images.  
        """  
  
        # Create a 2x5 grid for displaying the images  
        fig, axes = plt.subplots(2, 5, figsize=(10, 5))  
        fig.suptitle("MNIST Sample Digits", fontsize=16)  
  
        # Find the unique classes and plot one example for each  
        unique_digits = np.unique(digits.target)  
        for ax, digit in zip(axes.ravel(), unique_digits):  
            # Get the index of the first occurrence of each digit  
            image_index = np.where(digits.target == digit)[0][0]  
            # Display the corresponding image  
            ax.imshow(digits.images[image_index], cmap='gray')  
            ax.set_title(f"Digit: {digit}")  
            ax.axis('off') # Turn off axes for cleaner visualization  
  
        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)
```

## 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 `kNNClassifier` (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 κλάσεις 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
        """
        self.model = LogisticRegression(solver=sol)

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

        Inputs:
        trainData: Training images (N,64)
        trainLabels: Labels (N,)
        """
        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,)  
    """  
    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  
        """  
        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,)  
        """  
        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,)
"""
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: 8.333333

```
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 modify k if desired however k=3 gave the best result
knnClassifierX.train(X_train, y_train) # Train the kNN Classifier

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

k-NN Classifier accuracy: 96.666667

```

C:\Users\Kyrillos\anaconda3\envs\askisi\Lib\site-packages\joblib\externals\loky\backend\context.py:136: UserWarning: Could not
find the number of physical cores for the following reason:
[WinError 2] The system cannot find the file specified
Returning the number of logical cores instead. You can silence this warning by setting LOKY_MAX_CPU_COUNT to the number of core
s you want to use.
  warnings.warn(
    File "C:\Users\Kyrillos\anaconda3\envs\askisi\Lib\site-packages\joblib\externals\loky\backend\context.py", line 257, in _count_physical_cores
      cpu_info = subprocess.run(
                  ^^^^^^^^^^^^^^^^^
    File "C:\Users\Kyrillos\anaconda3\envs\askisi\Lib\subprocess.py", line 548, in run
      with Popen(*popenargs, **kwargs) as process:
          ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    File "C:\Users\Kyrillos\anaconda3\envs\askisi\Lib\subprocess.py", line 1026, in __init__
      self._execute_child(args, executable, preexec_fn, close_fds,
    File "C:\Users\Kyrillos\anaconda3\envs\askisi\Lib\subprocess.py", line 1538, in _execute_child
      hp, ht, pid, tid = _winapi.CreateProcess(executable, args,
                          ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

```

## 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
import numpy as np
import matplotlib.pyplot as plt

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

    Parameters:
    - testData: Test dataset
    - testLabels: Ground truth labels
    - classifier: The classifier to evaluate

    Returns:
    - M: Normalized confusion matrix
    - acc: Overall accuracy
    """
    batchSize = 50
    num_classes = 10 # For MNIST, we have 10 classes
    M = np.zeros((num_classes, num_classes), dtype=np.int32)

    # Process test data in batches
    for i in range(0, testData.shape[0], batchSize):
        batch_X = testData[i:i + batchSize]
        batch_y = testLabels[i:i + batchSize]
        predictions = classifier(batch_X)

        # Update confusion matrix
        for true_label, predicted_label in zip(batch_y, predictions):
            M[true_label, predicted_label] += 1

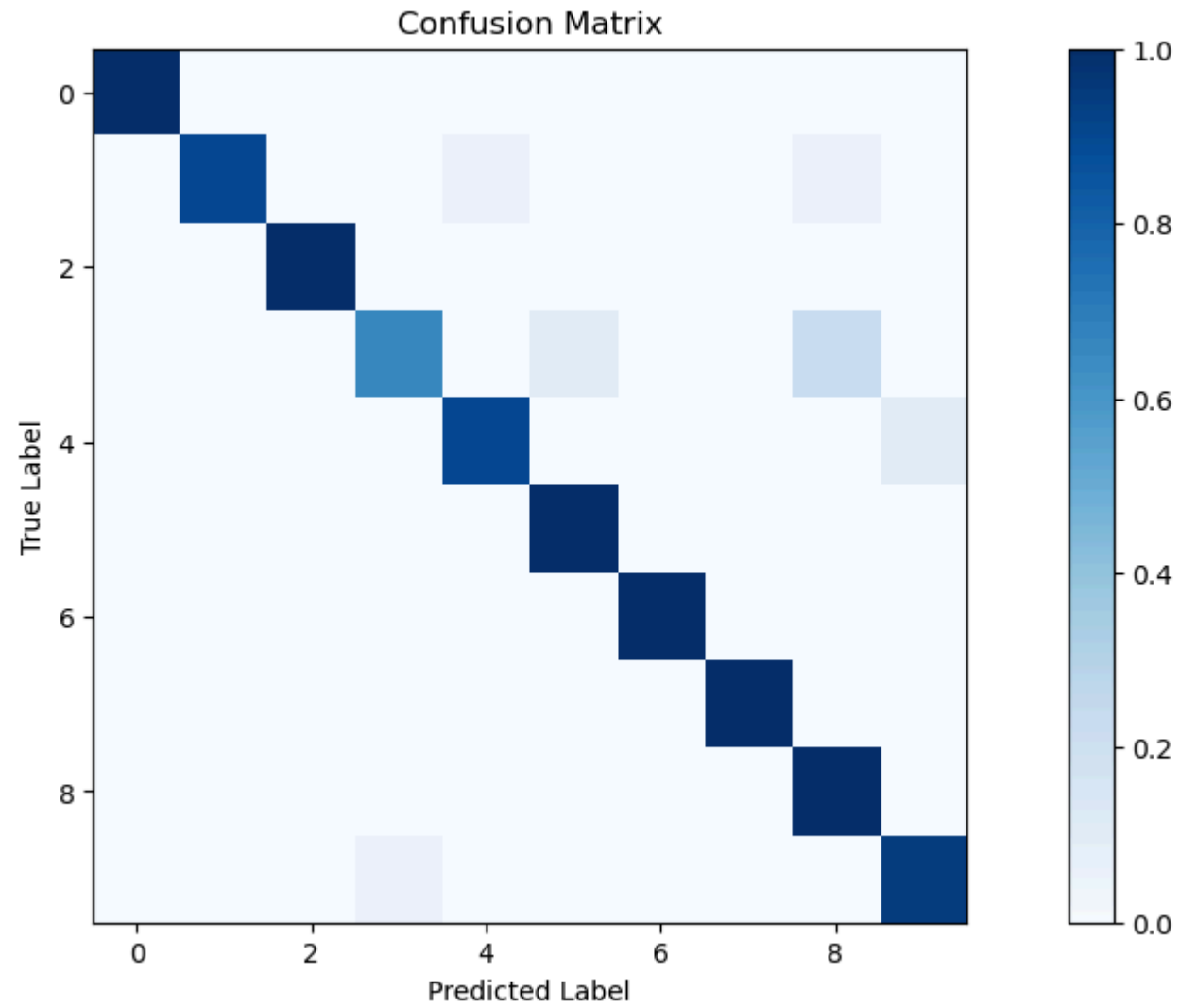
    # Calculate accuracy
    correct_predictions = np.trace(M)
    total_predictions = np.sum(M)
    acc = correct_predictions / total_predictions if total_predictions > 0 else 0.0

    # Normalize confusion matrix rows to sum to 1
    row_sums = M.sum(axis=1, keepdims=True)
    M = np.divide(M, row_sums, out=np.zeros_like(M, dtype=np.float32), where=row_sums != 0)
```

```
return M, acc
```

```
def VisualizeConfussion(M):  
    plt.figure(figsize=(14, 6))  
    plt.imshow(M, interpolation='nearest', cmap='Blues')  
    plt.colorbar()  
    plt.xlabel('Predicted Label')  
    plt.ylabel('True Label')  
    plt.title('Confusion Matrix')  
    plt.show()  
    print("Normalized Confusion Matrix:")  
    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)
```



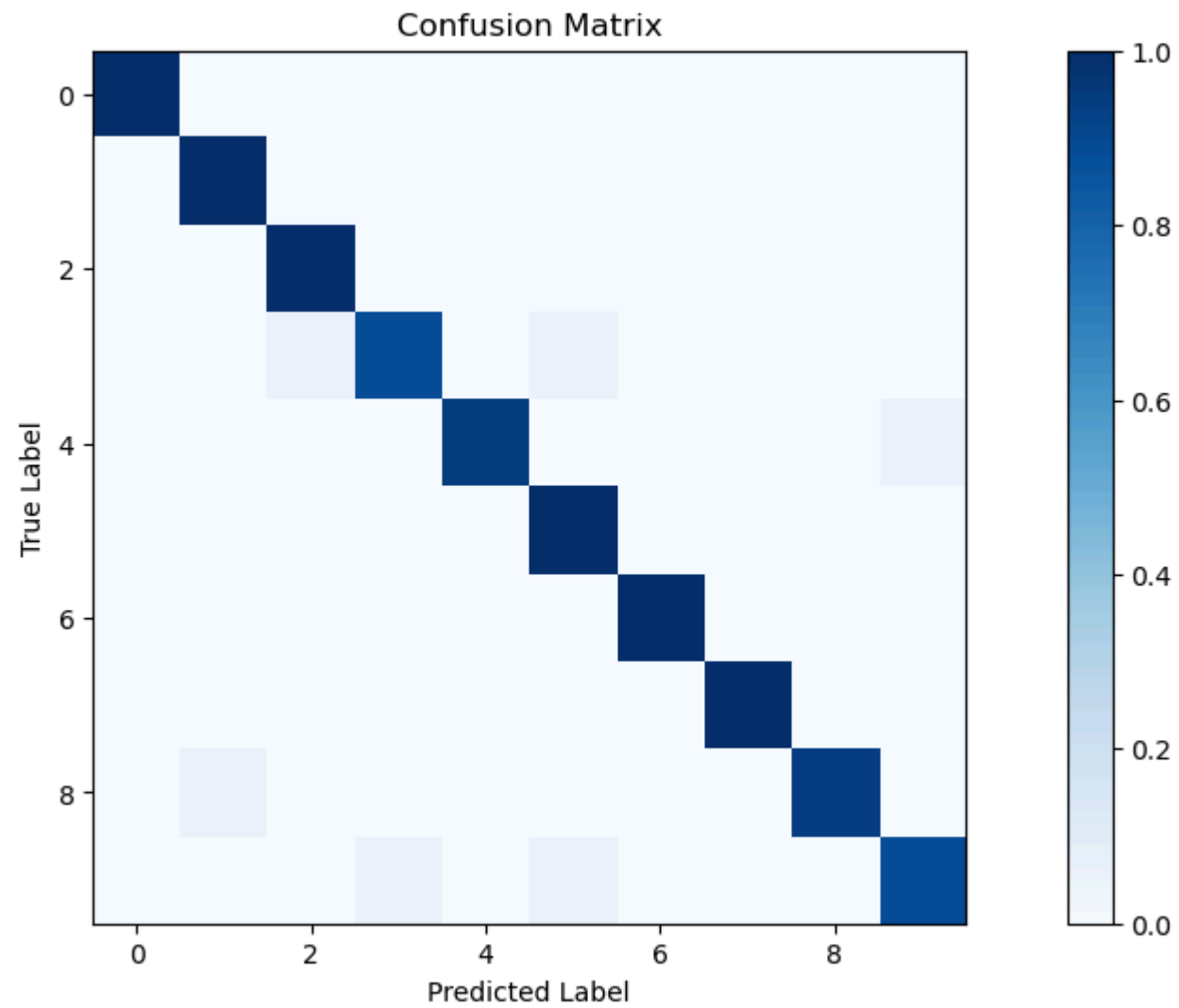


Normalized Confusion Matrix:

```
[[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)
```



Normalized Confusion Matrix:

```
[[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]: 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."""
        return np.sqrt(np.sum((x1 - x2) ** 2))

    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 = []
# Calculate distance from x to all training points
for i in range(self.X_train.shape[0]):
    distance = self.euclidean_distance(x, self.X_train[i])
    distances.append((distance, i))

# Sort by distance
distances.sort(key=lambda pair: pair[0])

# Select the first k indices
indices = [pair[1] for pair in distances[:self.k]]
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 x in X:
        # Find the k nearest neighbors
        neighbors = self.find_k_nearest_neighbors(x)

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

        # "Vote" for the most frequent label
        predicted_label = max(set(neighbor_labels), key=neighbor_labels.count)
        predicted_labels.append(predicted_label)

    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: 95.555556

## 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]: import numpy as np

def svd(A):
    """
    Computes the Singular Value Decomposition (SVD) of a matrix A.
    Returns U, singular_values, V.T.
    """
    # Compute the covariance matrix
```

```
covariance_matrix = np.dot(A.T, A)

# Eigen decomposition
eigenvalues, eigenvectors = np.linalg.eigh(covariance_matrix)

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

# Handle negative or zero eigenvalues
eigenvalues = np.where(eigenvalues > 0, eigenvalues, 1e-10)

# Compute singular values
singular_values = np.sqrt(eigenvalues)

# Compute U matrix
V = eigenvectors
U = np.dot(A, V) / singular_values

return U, singular_values, V.T


class PCAKNNClassifier():
    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
        self.k = k
        self.knn = kNNClassifier_v1_5(k=self.k) # Use previous kNN implementation

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

```

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

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

# Perform SVD on centered data (mean-deviation form of data matrix)
U, D, V = svd(X_hat)

# Keep only the top components
self.V = V[:self.components]

# Project the training data into the PCA space
self.X_train_pca = np.dot(X_hat, self.V.T)
self.y_train = trainLabels

# Train the kNN classifier in PCA space
self.knn.train(self.X_train_pca, self.y_train)

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

    Inputs:
    X: Test images (N,64)

    Returns:
    predicted labels (N,)
    """

    # Project the test data into the PCA space
    X_hat = X - self.mean
    X_pca = np.dot(X_hat, self.V.T)

    # Use the kNN classifier to predict
    return self.knn(X_pca)

# test your classifier with only the first 100 training examples (use this
# while debugging)

```

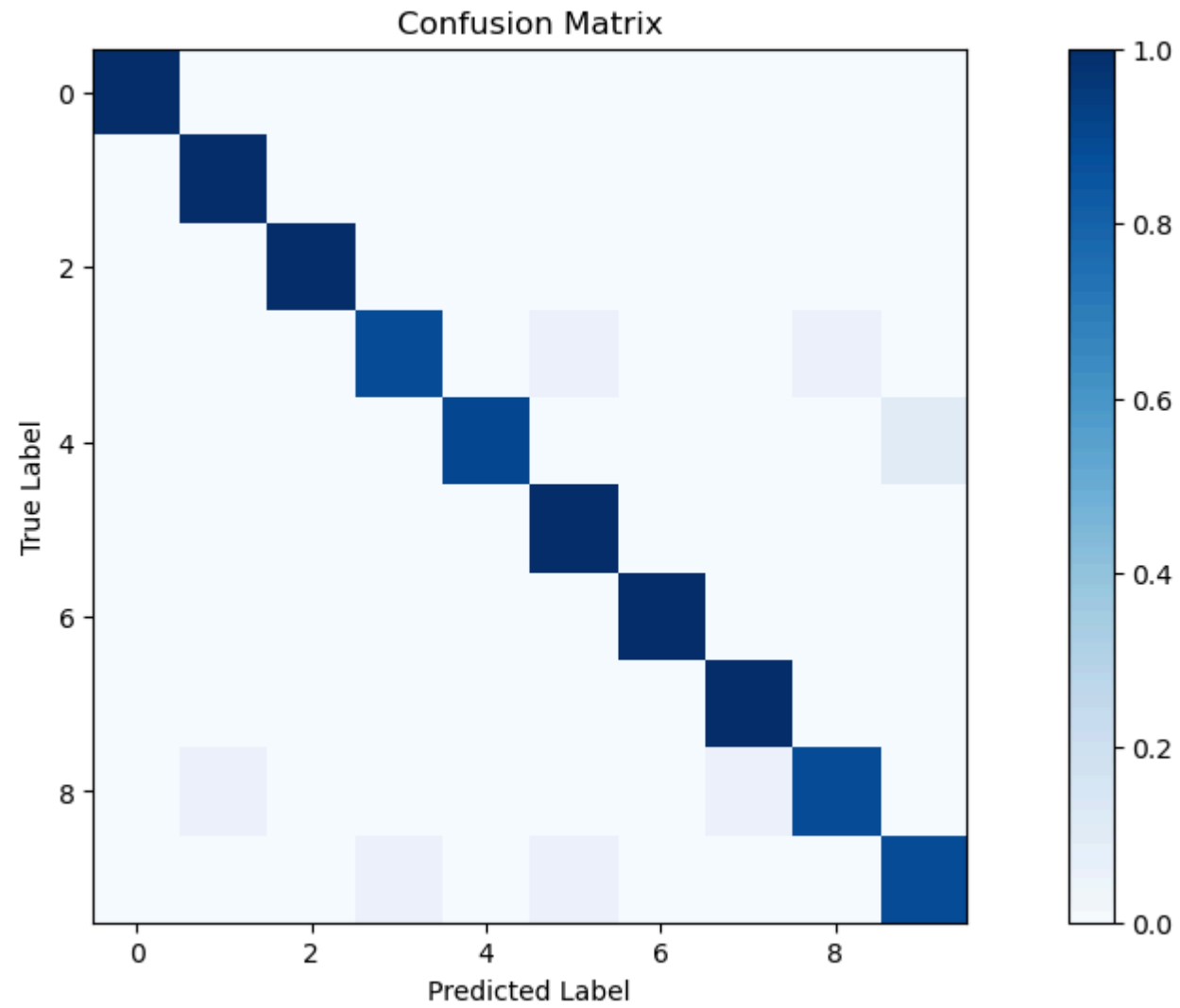
```
pcaknnClassifierX = PCAKNNClassifier()  
pcaknnClassifierX.train(X_train[:100], y_train[:100])  
print('PCA-kNN classifier accuracy (100 samples): %f' % test(X_test, y_test, pcaknnClassifierX))
```

PCA-kNN classifier accuracy (100 samples): 84.444444

```
In [18]: # test your classifier with all the training examples  
pcaknnClassifier = PCAKNNClassifier()  
pcaknnClassifier.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, pcaknnClassifier)  
  
# Display the accuracy and visualize the confusion matrix  
print ('PCA-kNN classifier accuracy: %f'%test(X_test, y_test, pcaknnClassifier))  
VisualizeConfussion(M_pca)
```

PCA-kNN classifier accuracy: 95.555556





Normalized Confusion Matrix:

```
[[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.  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.  0.06 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 κλασικός k-NN ταξινομητής πέτυχε ακρίβεια 95.55%, δουλεύοντας απευθείας στον χώρο των pixels (64 διαστάσεις). Η υψηλή αυτή ακρίβεια είναι αναμενόμενη, καθώς ο ταξινομητής έχει πλήρη πρόσβαση σε όλες τις διαθέσιμες πληροφορίες από τα δεδομένα. Αντίθετα, όταν χρησιμοποιήθηκε ο PCA-kNN ταξινομητής με εκπαίδευση σε μόλις 100 δείγματα, η ακρίβεια μειώθηκε στο 84.44%. Αυτό οφείλεται στο γεγονός ότι ο αριθμός των δειγμάτων δεν επαρκεί για να εκτιμηθούν κατάλληλα οι 25 συνιστώσες μέσω της ανάλυσης PCA, με αποτέλεσμα να χάνονται σημαντικές πληροφορίες κατά τη μείωση των διαστάσεων.

Όταν, όμως, ο PCA-kNN εκπαιδεύτηκε σε ολόκληρο το σύνολο δεδομένων, η ακρίβεια ανέβηκε στο 95.55%, φτάνοντας ακριβώς το ίδιο επίπεδο με τον κλασικό k-NN. Αυτό δείχνει ότι η χρήση του PCA δεν επηρεάζει την απόδοση του ταξινομητή όταν τα δεδομένα εκπαίδευσης επαρκούν. Η ανάλυση σε 25 συνιστώσες καταφέρνει να διατηρήσει τις κρίσιμες πληροφορίες του αρχικού χώρου των 64 διαστάσεων, αποδεικνύοντας ότι η μείωση των διαστάσεων μπορεί να είναι εξαιρετικά αποτελεσματική.

Συγκρίνοντας τους δύο ταξινομητές, βλέπουμε ότι ο PCA-kNN έχει το πλεονέκτημα της βελτιωμένης αποδοτικότητας κατά τη φάση της πρόβλεψης, καθώς οι υπολογισμοί εκτελούνται σε έναν χώρο μικρότερων διαστάσεων (25 αντί για 64). Ωστόσο, είναι πιο ευαίσθητος στον αριθμό των δειγμάτων εκπαίδευσης, κάτι που φαίνεται από την πτώση της ακρίβειας στο 84.44% όταν εκπαιδεύεται σε λίγα δεδομένα. Από την άλλη πλευρά, ο κλασικός k-NN δεν απαιτεί καμία προεπεξεργασία ή μείωση διαστάσεων, καθιστώντας τον πιο απλό στη χρήση για μικρότερα σύνολα δεδομένων ή όταν ο χρόνος εκπαίδευσης δεν αποτελεί πρόβλημα.

Συνολικά, ο PCA-kNN είναι προτιμότερος για μεγαλύτερα σύνολα δεδομένων, καθώς μειώνει την υπολογιστική πολυπλοκότητα κατά τη φάση της πρόβλεψης, ενώ διατηρεί την απόδοσή του σε επίπεδα συγκρίσιμα με τον κλασικό k-NN, εφόσον τα δεδομένα εκπαίδευσης επαρκούν. Αντίθετα, ο κλασικός k-NN μπορεί να είναι καλύτερη επιλογή σε περιπτώσεις με μικρότερα σύνολα δεδομένων ή όταν δεν υπάρχει ανάγκη για μείωση των διαστάσεων.

## 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([[0.3358, 0.6984, 0.0724],
        [0.3579, 0.6247, 0.6564],
        [0.7530, 0.7776, 0.4112],
        [0.2968, 0.4307, 0.0623],
        [0.3757, 0.0331, 0.9693]])
```

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]: 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 = './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()
```

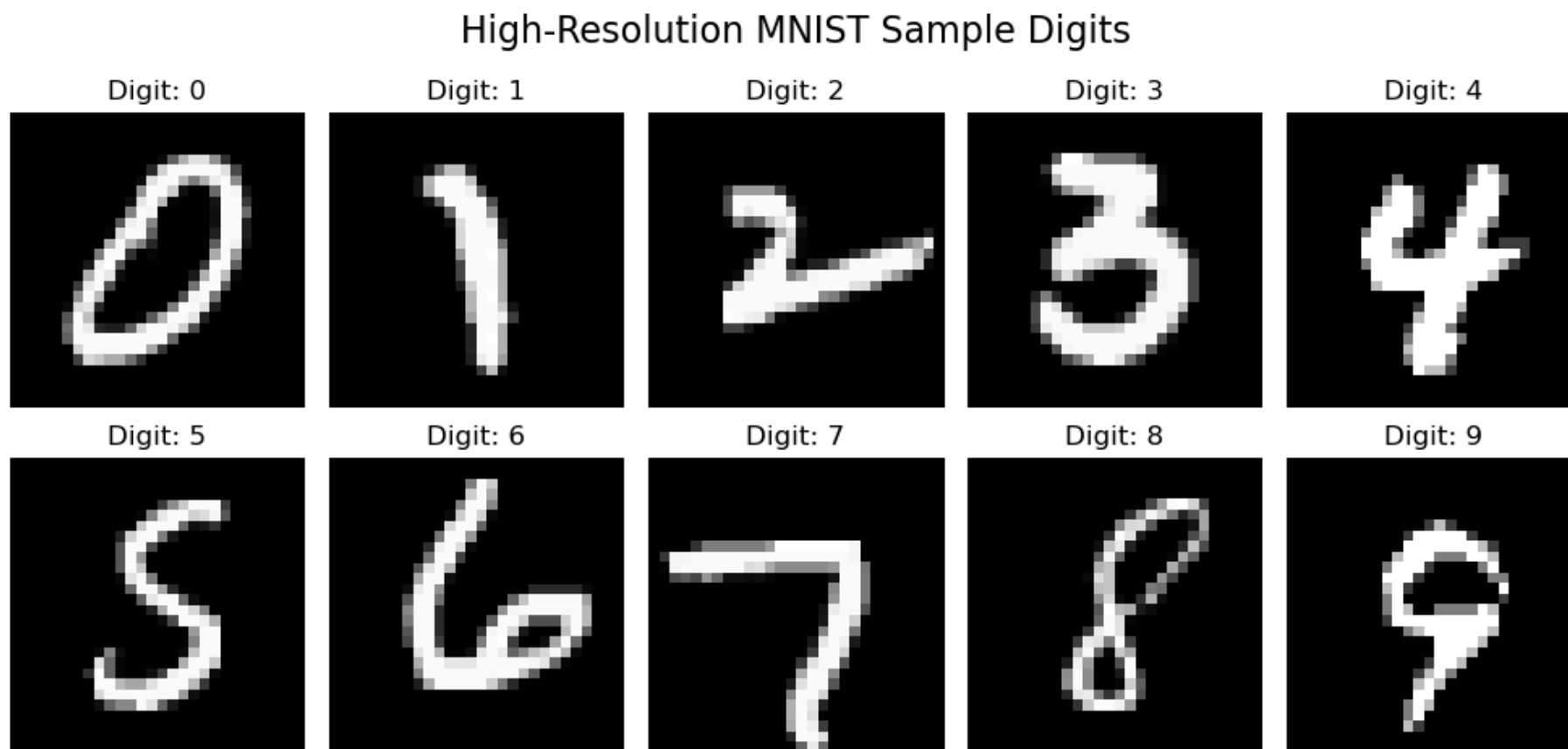
```
In [21]: def plot_mnist_sample_high_res(X_train, y_train):
        """
        This function plots a sample image for each category.
        The result is a figure with a 2x5 grid of images.
        """
        # Create a 2x5 grid for displaying the images
        fig, axes = plt.subplots(2, 5, figsize=(10, 5))
        fig.suptitle("High-Resolution MNIST Sample Digits", fontsize=16)

        # Loop through each digit (0-9)
        for ax, digit in zip(axes.ravel(), range(10)):
            # Find all indices of the current digit in the dataset
            indices = np.where(y_train == digit)[0]
            # Select a random index for the current digit
            random_index = np.random.choice(indices)
            # Display the corresponding image
            ax.imshow(X_train[random_index], cmap='gray')
            ax.set_title(f"Digit: {digit}")
            ax.axis('off')

        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/κλάση της DNN (βλ. παράδειγμα "LinearClassifier(DNN)"). Μπορείτε να ανατρέξετε στη διεύθυνση [https://pytorch.org/tutorials/beginner/pytorch\\_with\\_examples.html](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]: import torch
import torch.nn as nn
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
import numpy as np

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):
        """
        Train the neural network.
        Args:
            X_train: Training images (input data).
            y_train: Training labels.
```

```
        epochs: Number of training epochs.
        batchSize: Size of each batch.
    """
    # Select the Loss function
    criterion = nn.CrossEntropyLoss()

    # Select the optimizer
    optimizer = optim.Adam(self.parameters(), lr=0.001, weight_decay=1e-4)

    # Convert the data to PyTorch tensors
    X_train = torch.FloatTensor(X_train)
    y_train = torch.LongTensor(y_train)

    # Move to GPU if available
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    self.to(device)
    X_train, y_train = X_train.to(device), y_train.to(device)

    # Training Loop
    for epoch in range(epochs):
        running_loss = 0.0
        # Create batches
        for i in tqdm(range(0, len(X_train), batchSize), desc=f"Epoch {epoch+1}/{epochs}"):
            # Get the current batch
            inputs = X_train[i:i+batchSize]
            labels = y_train[i:i+batchSize]

            # Wrap inputs and labels in Variable
            inputs, labels = Variable(inputs), Variable(labels)

            # Zero the parameter gradients
            optimizer.zero_grad()

            # Forward pass
            outputs = self.forward(inputs)

            # Compute Loss
            loss = criterion(outputs, labels)

            # Backward pass
            loss.backward()
```



```

        # Optimize weights
        optimizer.step()

        # Accumulate Loss
        running_loss += loss.item()

    # Print average loss for the epoch
    print(f"Epoch {epoch+1}/{epochs}, Loss: {running_loss / len(X_train):.4f}")

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):
        # Flatten input to [batch_size, input_features]
        x = x.view(x.size(0), -1)

```

```
# linear operation
return torch.addmm(self.bias1, x, self.weight1.t())

#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))

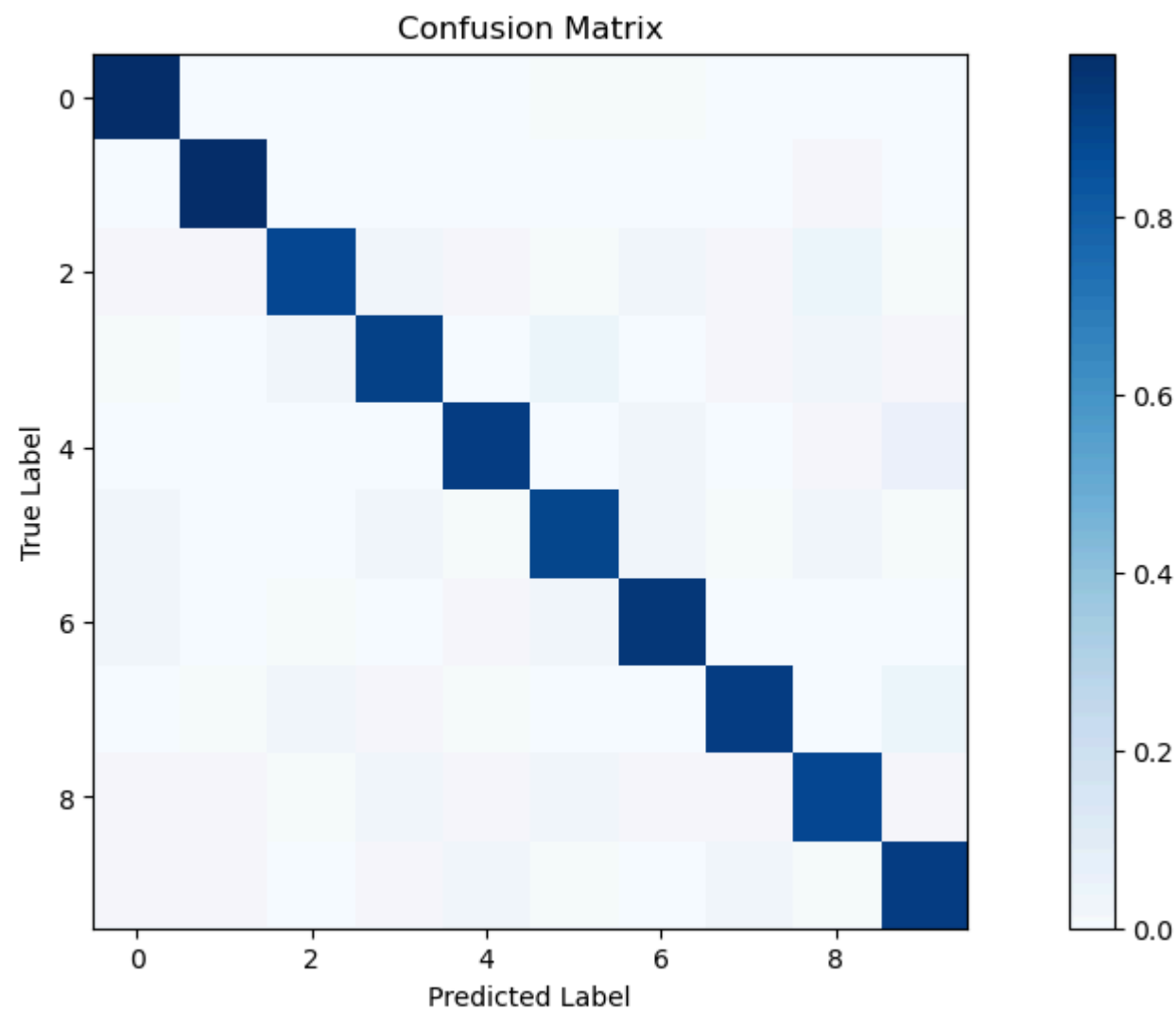
## In case abovementioned 4 lines return error: Modify the lines for transposing X_train
## and X_test by uncommenting the following 4 lines and place the 4 lines above in comments

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)
```

```
Epoch 7/10: 100%|██████████████████████████████████████████████████████████████████████████| 1200/1200 [00:00<00:00, 3586.94it/s]
Epoch 7/10, Loss: 0.0054
Epoch 8/10: 100%|██████████████████████████████████████████████████████████████████████████| 1200/1200 [00:00<00:00, 3562.35it/s]
Epoch 8/10, Loss: 0.0054
Epoch 9/10: 100%|██████████████████████████████████████████████████████████████████████████| 1200/1200 [00:00<00:00, 3638.55it/s]
Epoch 9/10, Loss: 0.0053
Epoch 10/10: 100%|██████████████████████████████████████████████████████████████████████████| 1200/1200 [00:00<00:00, 3506.88it/s]
Epoch 10/10, Loss: 0.0053
Linear classifier accuracy: 92.490000
```

```
In [26]: # display confusion matrix
M_linear, acc_linear = Confusion(X_test, y_test, linearClassifier)
VisualizeConfussion(M_linear)
```



Normalized Confusion Matrix:

```
[[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 0.88 0.02 0.01 0. 0.01 0.01 0.04 0.01]
 [0. 0. 0.02 0.9 0. 0.03 0. 0.01 0.02 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 0.01 0.01 0.03 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. 0.04]
 [0.01 0.01 0.01 0.02 0.01 0.03 0.01 0.01 0.88 0.01]
 [0.01 0.01 0. 0.01 0.02 0.01 0. 0.02 0. 0.92]]
```

## 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]: import matplotlib.pyplot as plt
import numpy as np

def visualize_weights(model):
    """
    Visualize the weights of the LinearClassifier as images.
    Args:
        model: The trained LinearClassifier.
    """
    # Extract weights from the model (Shape: [10, 784])
    weights = model.weight1.data.cpu().numpy()

    # Normalize weights between 0 and 1 for better visualization
    normalized_weights = (weights - weights.min()) / (weights.max() - weights.min())
```

```
# Create a plot for each class
fig, axes = plt.subplots(2, 5, figsize=(12, 6))
fig.suptitle("Visualization of Weights for Each Class", fontsize=16)

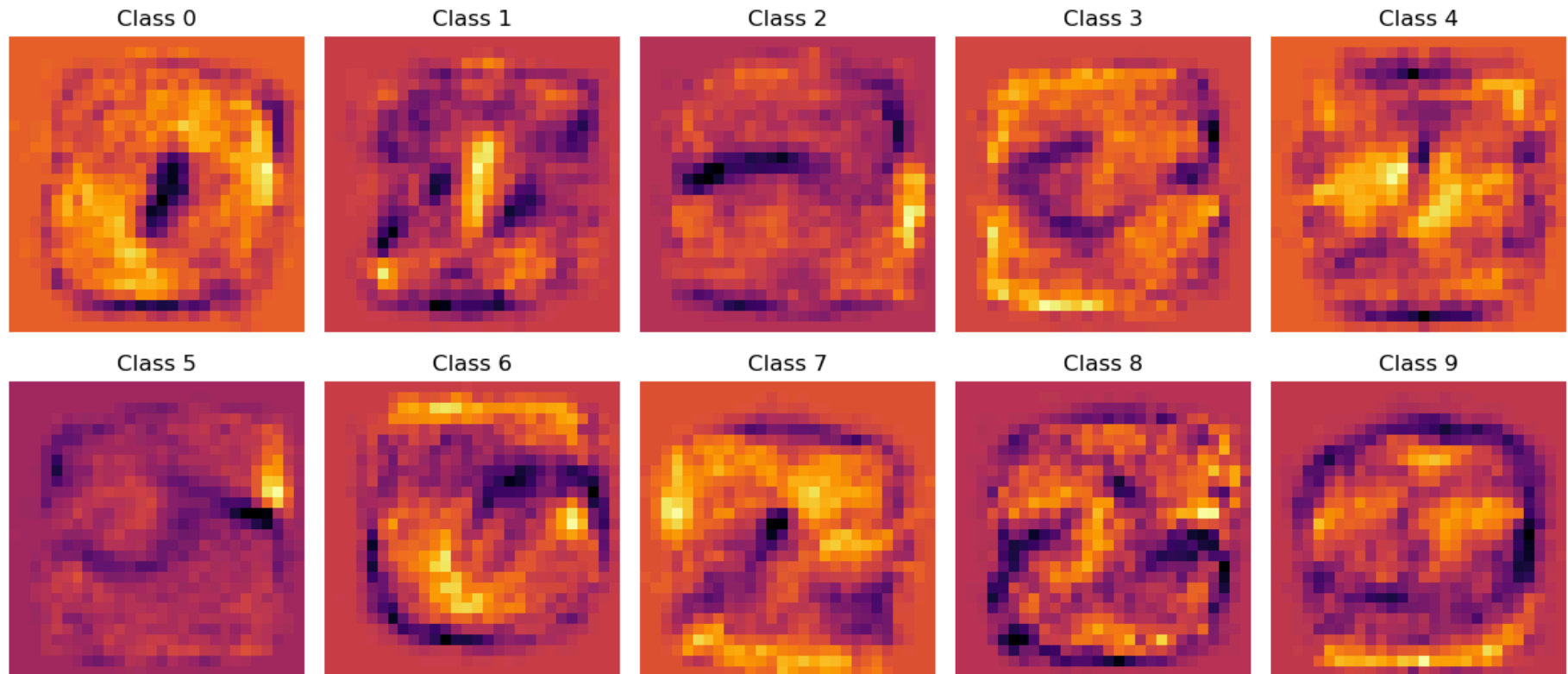
for i, ax in enumerate(axes.ravel()):
    # Reshape weights for class `i` to 28x28 (MNIST dimensions)
    weight_image = normalized_weights[i].reshape(28, 28)

    # Display the image with a colormap (e.g., 'inferno')
    ax.imshow(weight_image, cmap='inferno')
    ax.set_title(f"Class {i}")
    ax.axis('off') # Turn off axes for better visualization

plt.tight_layout()
plt.show()

# Call the function for the trained LinearClassifier
visualize_weights(linearClassifier)
```

## Visualization of Weights for Each Class



### Σχολιασμός των βαρών

Τα βάρη που απεικονίζονται από τον γραμμικό ταξινομητή μοιάζουν με θολές εκδοχές των ψηφίων του MNIST, καθώς κάθε φίλτρο αντιπροσωπεύει τα χαρακτηριστικά που είναι σημαντικά για την αναγνώριση κάθε κατηγορίας. Αυτό συμβαίνει επειδή ο ταξινομητής μαθαίνει να ενισχύει περιοχές της εισόδου που σχετίζονται με συγκεκριμένα χαρακτηριστικά του ψηφίου. Επιπλέον, παρατηρείται ότι ψηφία με παρόμοια σχήματα, όπως το 8 και το 0 ή το 1 και το 7, έχουν φίλτρα που μοιάζουν μεταξύ τους, καθώς μοιράζονται κοινά χαρακτηριστικά. Οι θολές περιοχές στα φίλτρα δείχνουν ότι ο γραμμικός ταξινομητής βασίζεται σε ευρύτερες περιοχές του ψηφίου για να πάρει αποφάσεις, χωρίς να εστιάζει σε τοπικές λεπτομέρειες. Τέλος, ορισμένα φίλτρα μπορεί να παρουσιάζουν ασάφειες ή παραμορφώσεις, ειδικά σε περιπτώσεις που οι κατηγορίες είναι πιο δύσκολο να διαχωριστούν, όπως τα ψηφία 3 και 5. Αυτά τα χαρακτηριστικά αντανακλούν τους περιορισμούς του γραμμικού ταξινομητή, ο οποίος βασίζεται σε γραμμικούς συνδυασμούς των pixels και δεν έχει την ικανότητα να ανιχνεύσει πιο πολύπλοκα τοπικά χαρακτηριστικά.

## 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 MLPClassifier(DNN):
          def __init__(self, in_features=28*28, classes=10, hidden=50):
              """
              Initialize weight and bias variables
              """
```





```
Epoch 10/10: 100%|██████████████████████████████████████████████████████████████████████████████| 1200/1200 [00:00<00:00, 2366.03it/s]
Epoch 10/10, Loss: 0.0014
```

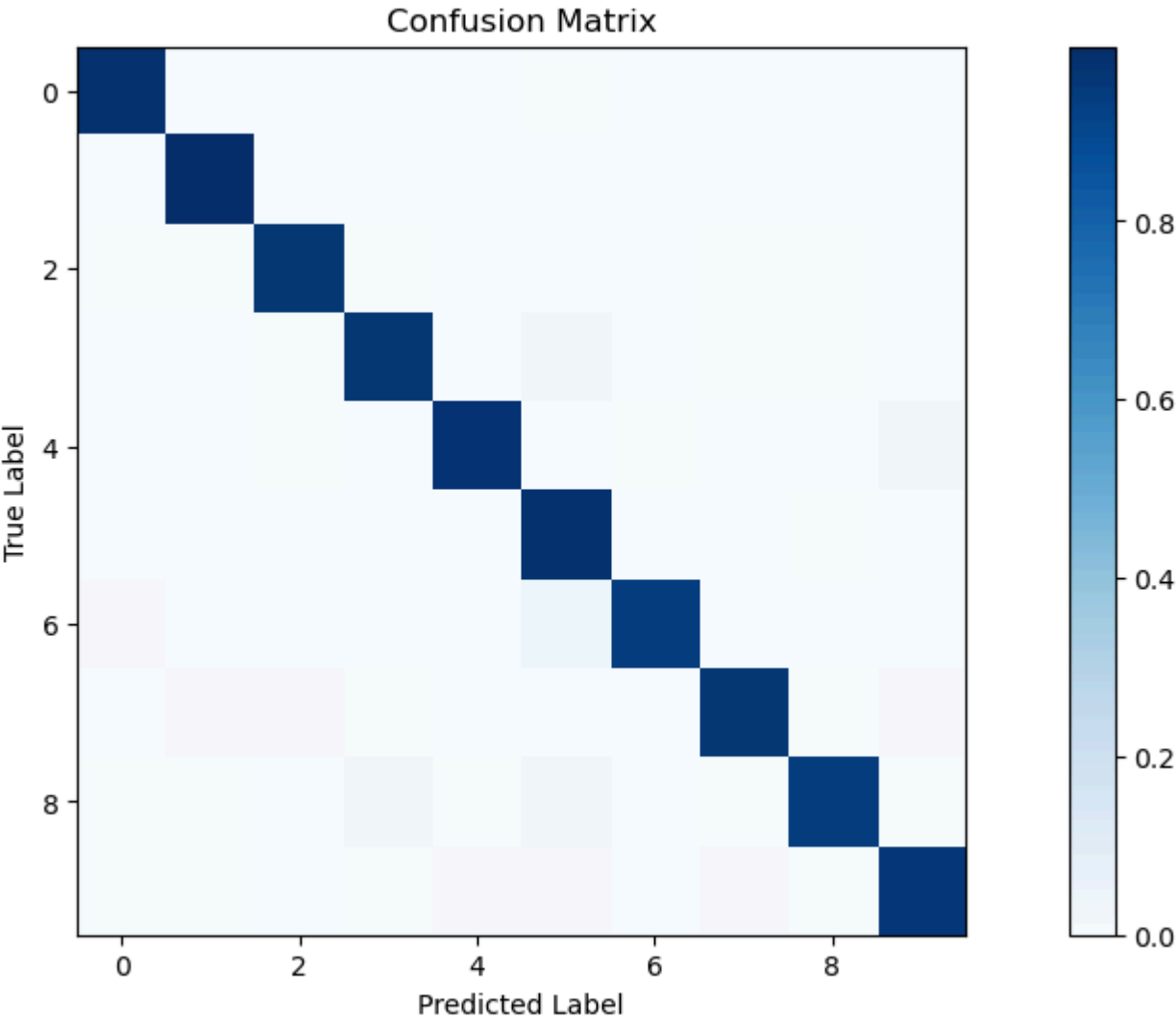
```
In [29]: # Plot confusion matrix
M_mlp, acc_mlp = Confusion(X_test, y_test, mlpClassifier)

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, mlpClassifier))

VisualizeConfussion(M_mlp)
```

```
Confusion matrix - MLP classifier accuracy: 0.966400
MLP classifier accuracy: 96.640000
```

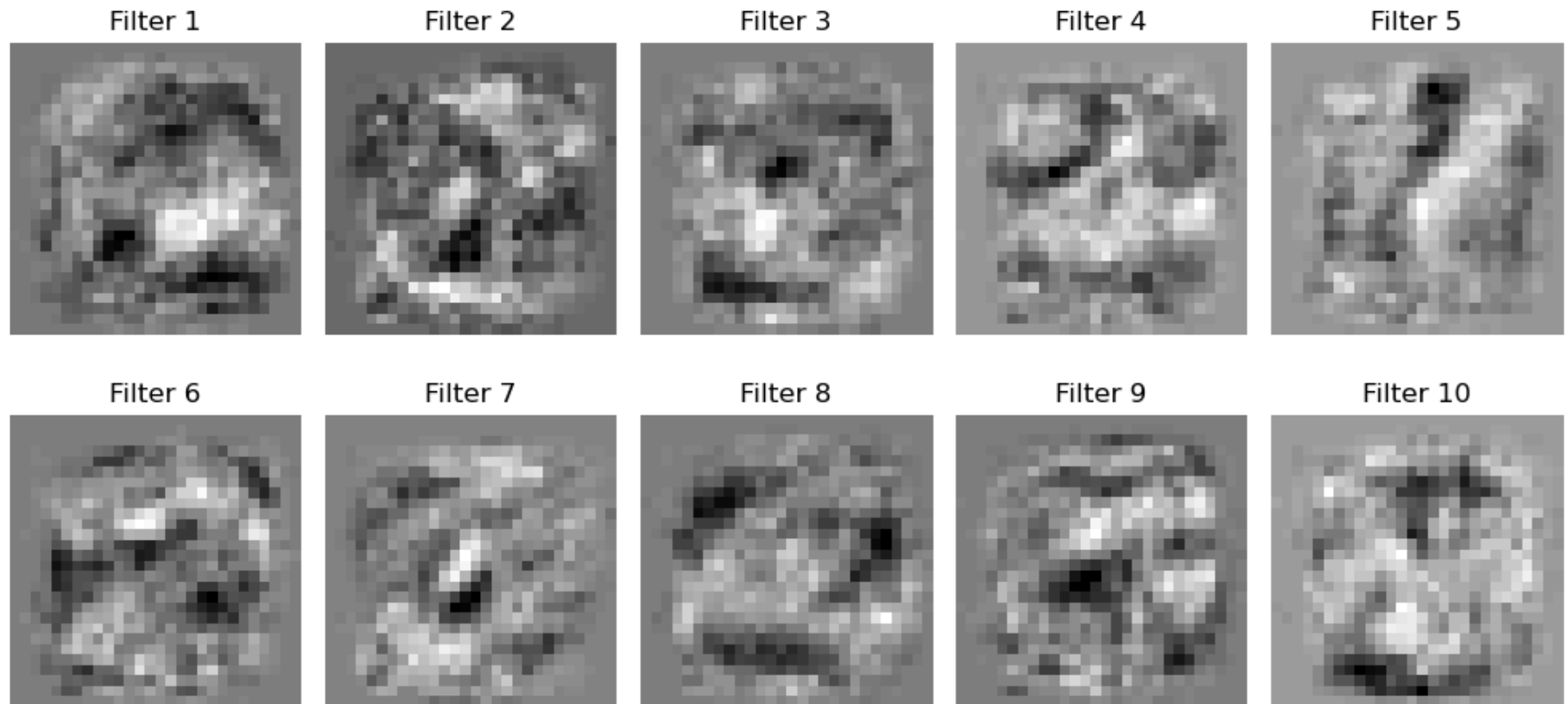


Normalized Confusion Matrix:

```
[[0.98 0.  0.  0.  0.  0.01 0.  0.  0.  0. ]
 [0.  1.  0.  0.  0.  0.  0.  0.  0.  0. ]
 [0.01 0.01 0.96 0.  0.  0.  0.  0.01 0.01 0. ]
 [0.  0.  0.  0.96 0.  0.02 0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.97 0.  0.01 0.  0.  0.01]
 [0.  0.  0.  0.  0.  0.98 0.  0.  0.01 0. ]
 [0.01 0.  0.  0.  0.  0.04 0.94 0.  0.  0. ]
 [0.  0.01 0.01 0.01 0.  0.  0.  0.96 0.  0.01]
 [0.  0.01 0.  0.01 0.  0.02 0.  0.  0.94 0.01]
 [0.  0.01 0.  0.01 0.01 0.01 0.  0.01 0.01 0.95]]
```

In [30]: *# Plot filter weights*

```
weights = mlpClassifier.weight1.cpu().detach().numpy()
plt.figure(figsize=(10, 5))
for i in range(10): # Plot the first 10 filters
    plt.subplot(2, 5, i + 1)
    plt.imshow(weights[i].reshape(28, 28), cmap='gray')
    plt.title(f'Filter {i+1}')
    plt.axis('off')
plt.tight_layout()
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)
- $\text{ReLU}(\text{Conv}(\text{kernel\_size}=5 \times 5, \text{stride}=2, \text{output\_features}=n))$
- $\text{ReLU}(\text{Conv}(\text{kernel\_size}=5 \times 5, \text{stride}=2, \text{output\_features}=n \times 2))$
- $\text{ReLU}(\text{Linear}(\text{hidden units} = 64))$
- $\text{Linear}(\text{output\_features}=\text{classes})$

Δηλαδή, 2 συνελκτικά 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 CNNClassifier(DNN):
    def __init__(self, classes=10, n=10):
        super(CNNClassifier, self).__init__()
        # Initialize weights and biases for first convolutional layer
        self.W1 = weight_variable((n, 1, 5, 5)) # 1 input channel, n output channels, 5x5 kernel
        self.b1 = bias_variable((n))

        # Initialize weights and biases for second convolutional layer
        self.W2 = weight_variable((n * 2, n, 5, 5)) # n input channels, 2n output channels, 5x5 kernel
        self.b2 = bias_variable((n * 2))

        # Fully connected layer weights and biases
        self.W3 = weight_variable((64, (7 * 7 * n * 2))) # Flattened size after two conv layers
        self.b3 = bias_variable((64))
```

```

# Output layer weights and biases
self.W4 = weight_variable((classes, 64))
self.b4 = bias_variable((classes))

def forward(self, x):
    # First convolutional layer with ReLU and stride 2
    x = conv2d(x, self.W1, stride=2, bias=self.b1)
    x = F.relu(x)

    # Second convolutional layer with ReLU and stride 2
    x = conv2d(x, self.W2, stride=2, bias=self.b2)
    x = F.relu(x)

    # Flatten the output from the second conv layer
    x = x.view(x.size(0), -1)

    # Fully connected hidden layer with ReLU
    x = F.linear(x, self.W3, self.b3)
    x = F.relu(x)

    # Output Layer (logits)
    y = F.linear(x, self.W4, self.b4)
    return y

cnnClassifier = CNNClassifier()
cnnClassifier.train net(X_train, y_train, epochs=10, batchSize=50)

```

Epoch 1/10: 100%	<div><div></div></div>	1200/1200	[00:01<00:00, 703.66it/s]
Epoch 1/10, Loss: 0.0083			
Epoch 2/10: 100%	<div><div></div></div>	1200/1200	[00:01<00:00, 703.43it/s]
Epoch 2/10, Loss: 0.0022			
Epoch 3/10: 100%	<div><div></div></div>	1200/1200	[00:01<00:00, 711.49it/s]
Epoch 3/10, Loss: 0.0015			
Epoch 4/10: 100%	<div><div></div></div>	1200/1200	[00:01<00:00, 711.82it/s]
Epoch 4/10, Loss: 0.0012			
Epoch 5/10: 100%	<div><div></div></div>	1200/1200	[00:01<00:00, 704.54it/s]
Epoch 5/10, Loss: 0.0009			
Epoch 6/10: 100%	<div><div></div></div>	1200/1200	[00:01<00:00, 702.08it/s]
Epoch 6/10, Loss: 0.0008			

Epoch 7/10: 100%	<div><div></div></div>	1200/1200	[00:01<00:00, 692.80it/s]
Epoch 7/10, Loss: 0.0007			
Epoch 8/10: 100%	<div><div></div></div>	1200/1200	[00:01<00:00, 696.61it/s]
Epoch 8/10, Loss: 0.0006			
Epoch 9/10: 100%	<div><div></div></div>	1200/1200	[00:01<00:00, 714.21it/s]
Epoch 9/10, Loss: 0.0005			
Epoch 10/10: 100%	<div><div></div></div>	1200/1200	[00:01<00:00, 712.38it/s]
Epoch 10/10, Loss: 0.0004			

```
In [32]: # Plot confusion matrix and print the test accuracy of the classifier
M_cnn, acc_cnn = Confusion(X_test, y_test, cnnClassifier)

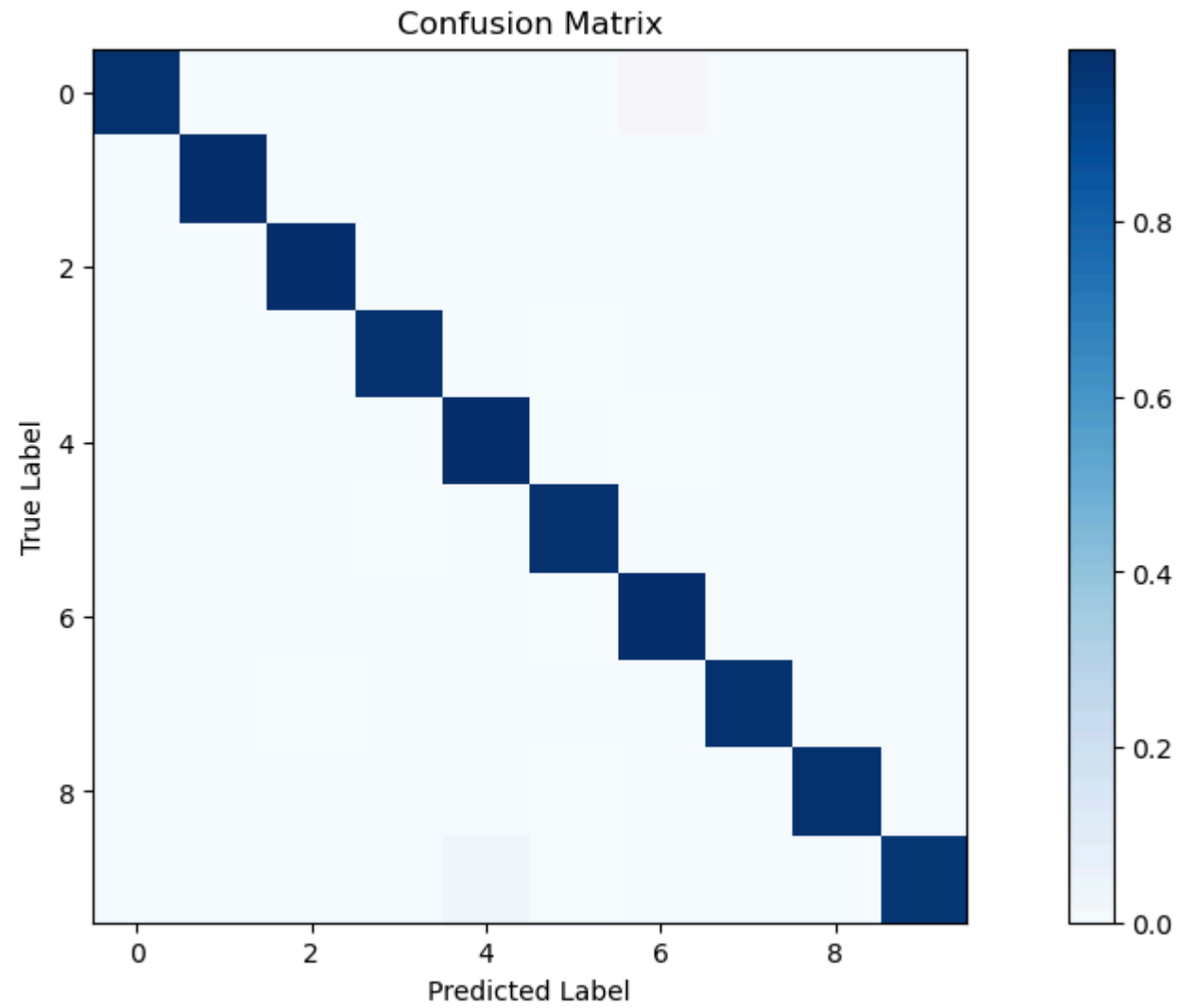
print ('Confusion matrix - CNN classifier accuracy: %f'%acc_cnn)

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

VisualizeConfusion(M_cnn)
```

```
Confusion matrix - CNN classifier accuracy: 0.987700
CNN classifier accuracy: 98.770000
```





Normalized Confusion Matrix:

```
[[0.99 0. 0. 0. 0. 0. 0.01 0. 0. 0. ]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. ]
 [0. 0. 0.99 0. 0. 0. 0. 0. 0. 0. ]
 [0. 0. 0. 0.99 0. 0. 0. 0. 0. 0. ]
 [0. 0. 0. 0. 0.99 0. 0.01 0. 0. 0. ]
 [0. 0. 0. 0.01 0. 0.99 0. 0. 0. 0. ]
 [0. 0. 0. 0. 0. 0.01 0.99 0. 0. 0. ]
 [0. 0. 0.01 0. 0. 0. 0. 0.99 0. 0. ]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.98 0. ]
 [0. 0. 0. 0. 0.02 0. 0. 0. 0. 0.97]]
```

- 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](https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html) καθώς και στο σχετικό tutorial στην αναρτημένη εργασία στη σελίδα course 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.