MYE046 – Υπολογιστική Όραση: Άνοιξη 2023

3η Σειρά Ασκήσεων: 50% του συνολικού βαθμού

Διδάσκων: Άγγελος Γιώτης

- ΠΑΡΑΔΟΣΗ: **Σάββατο, 10 Ιουνίου, 2023 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!
- 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.
- Οι απαντήσεις θα παραδοθούν με την εντολή: **turnin assignment_3@mye046 onoma.txt assignment3.ipynb assignment3.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 40 (από 50).

Άσκηση 1: Μηχανική Μάθηση [25 μονάδες]

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

Ζήτημα 1.1: Αρχική Εγκατάσταση

Θα χρησιμοποιήσουμε την ενότητα Scikit-learn (Sklearn) (https://scikit-learn.org/stable/) 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.

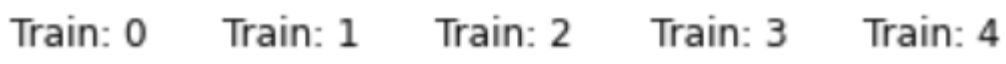
In [39]:
```
1 import sklearn
2 sklearn.__version__
```

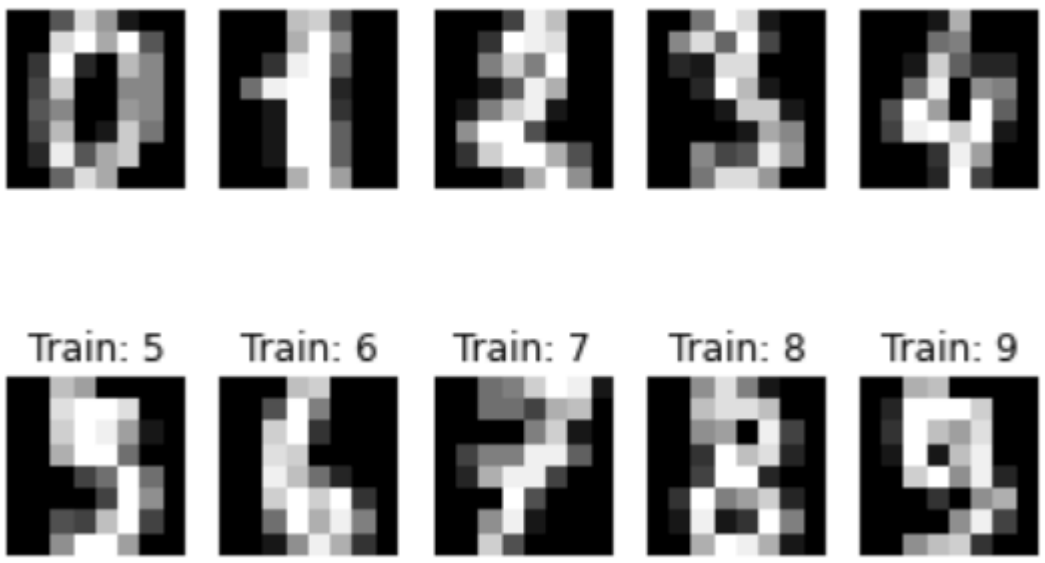
Out[39]:
```
'1.0.2'
```

Ζήτημα 1.2: Λήψη συνόλου δεδομένων χειρόγραφων ψηφίων "MNIST" και απεικόνιση παραδειγμάτων [2 μονάδες]

Η βάση δεδομένων MNIST (https://en.wikipedia.org/wiki/MNIST_database) (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:





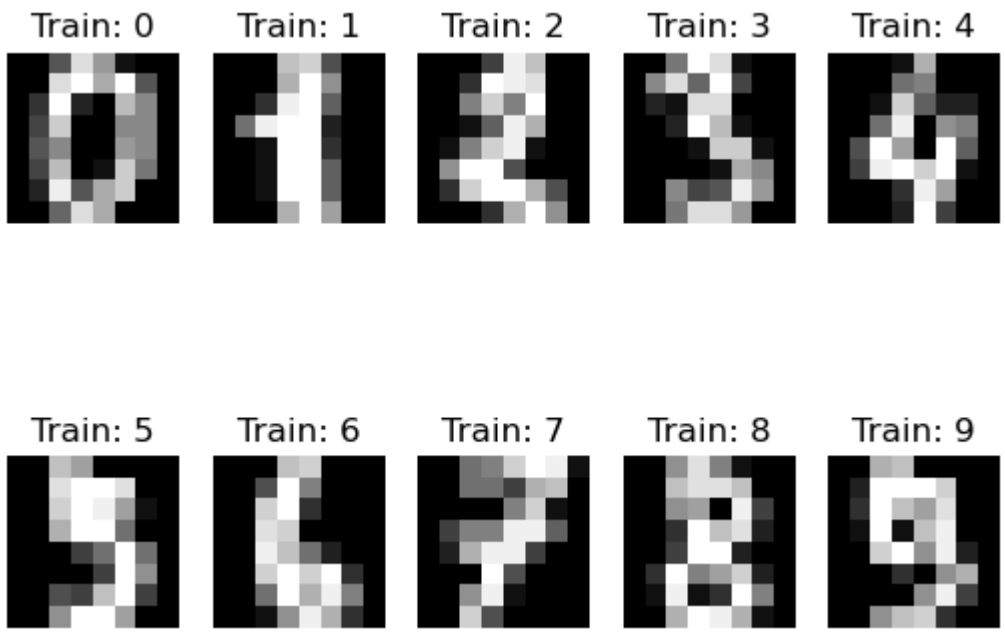
In [40]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
```

In [41]:
```
# 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 [42]:
```
def plot_mnist_sample(digits):
    """
    This function plots a sample image for each category,
    The result is a figure with 2x5 grid of images.

    """
    plt.figure()

    """ ==========
    YOUR CODE HERE
    ========== """
    for i in range(10): #ten images for numbers 0 thought 9
        plot = plt.subplot(2, 5, i + 1) #create a plot with size 2x5 and put it in position i+1
        #display the image of every digit
        #the image is filtered by the target label to equal to i
        #take the first image of the dataset of each digit, we could take any image
        #show the image in grayscale
        plot.imshow(digits.images[digits.target == i][0], cmap='gray')
        plot.set_title(f'Train: {i}') #set title to Train: i
        plot.axis('off') #remove axis

```

In [43]:
```
# 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 [4 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.

In [44]:
```
# 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 [45]:
```
# 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 [46]:

```python
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
        """
        """ ==========
        YOUR CODE HERE
        ========== """
        #create initialize LogisticRegression
        #using as solver method the liblinear
        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,)
        """
        """ ==========
        YOUR CODE HERE
        ========== """
        #fit the Logistic Regression model using the training data and training labels
        #Model fitting is the process of determining the coefficients b0, b1, …, br
        #that correspond to the best value of the cost function
        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,)
        """
        """ ==========
        YOUR CODE HERE
        ========== """
        #return a prediction on the test data using the trained Logistic Regression model
        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
        """
        """ ==========
        YOUR CODE HERE
        ========== """
        #create initialize KNeighborsClassifier
        #using 3 voting neighbors and computing them using brute force
        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,)
        """
        """ ==========
        YOUR CODE HERE
```

```
        ========== """
        #fit the KNeighborsClassifier model using the training data and training labels
        #so the model can learn from the 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,)
        """
        """ ==========
        YOUR CODE HERE
        ========== """
        #return a prediction on the test data using the trained KNeighborsClassifier model
        return self.model.predict(x)
```

In [47]:
```
# TEST CODE: DO NOT CHANGE
randomClassifierX = RandomClassifier()
print ('Random classifier accuracy: %f'%test(X_test, y_test, randomClassifierX))
```

```
Random classifier accuracy: 6.111111
```

In [48]:
```
# 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))
```

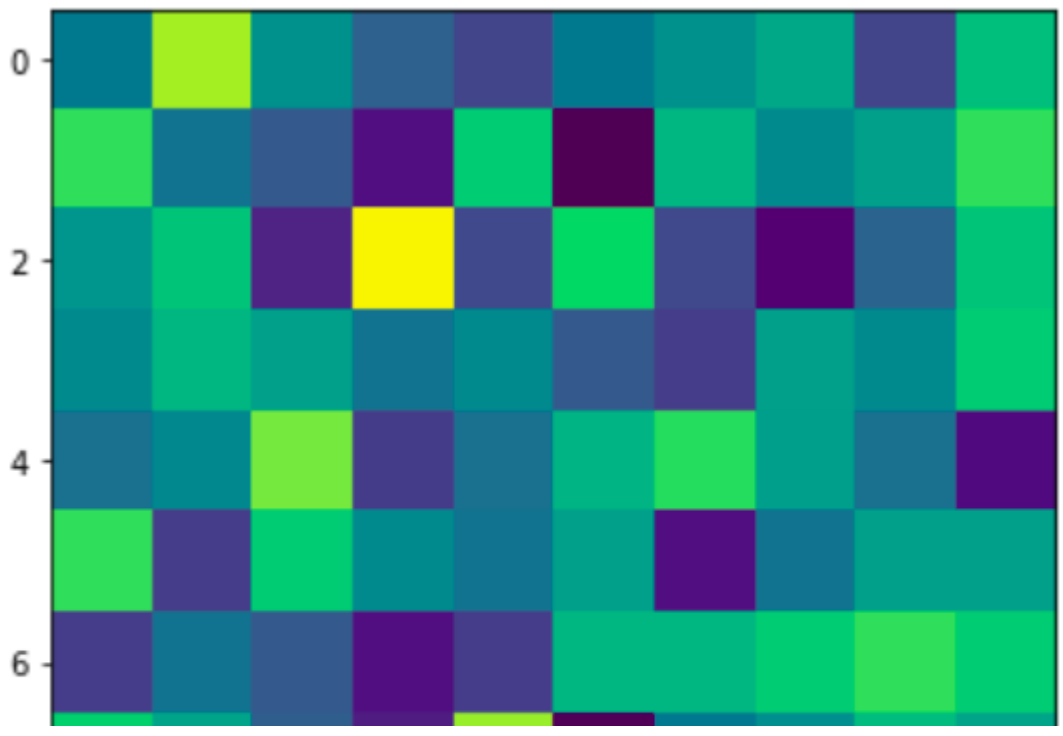
```
Logistic Regression Classifier classifier accuracy: 93.888889
```

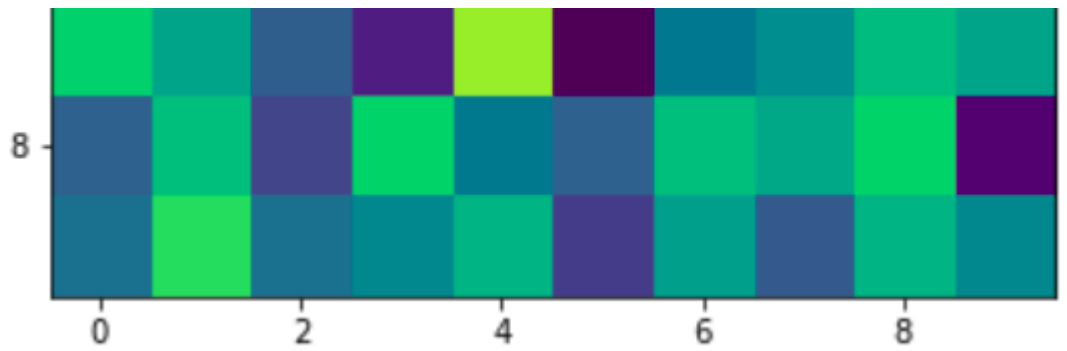
In [49]:
```
# TEST kNNClassifier
""" ==========
YOUR CODE HERE
========== """
knnClassifierX = kNNClassifier() #create an instance of the kNNClassifier
#train the knnClassifierX using X_train and y_train to train the 3-nearest neighbors
knnClassifierX.train(X_train, y_train)
#print the k-NN Classifier accuracy on the test data
#by comparing the predicted data with the true data
print('k-NN Classifier accuracy: %f' % test(X_test, y_test, knnClassifierX))

```

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

Ζήτημα 1.4: Πίνακας Σύγχυσης [4 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.



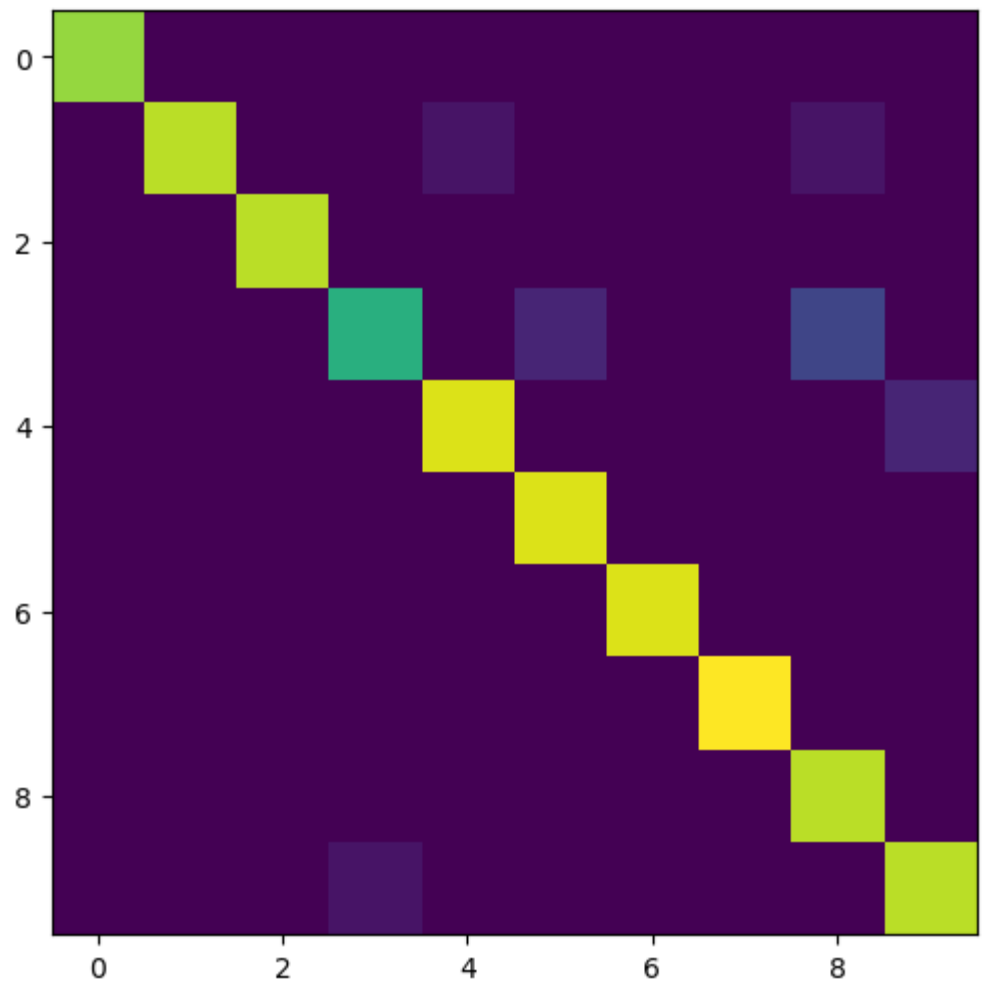


In [50]:

```
from tqdm import tqdm

def Confusion(testData, testLabels, classifier):
    batchsize=50 #number of samples in a batch
    correct=0
    M=np.zeros((10,10)) #confusion matrix
    num=testData.shape[0]/batchsize #number of batches
    count=0 #predicitons
    acc=0

    for data,label in tqdm(DataBatch(testData,testLabels,batchsize,shuffle=False),total=len(testData)//I
        """ ==========
        YOUR CODE HERE
        ========== """
        #for each batch of data make a prediction using the classifier
        predictions = classifier(data)
        #update the matrix M by increasing by 1 for each prediction
        for i in range(len(label)):
            M[label[i], predictions[i]] += 1
        #check by comparing the predicitons with the label and count correct predictions
        correct += np.sum(predictions == label)
        count += len(label) #total number of predictions

    # Calculate accuracy
    # acc = correct / count * 100.0
    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 [51]:
```
# TEST/PLOT CODE: DO NOT CHANGE
# TEST LogisticRegressionClassifier

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

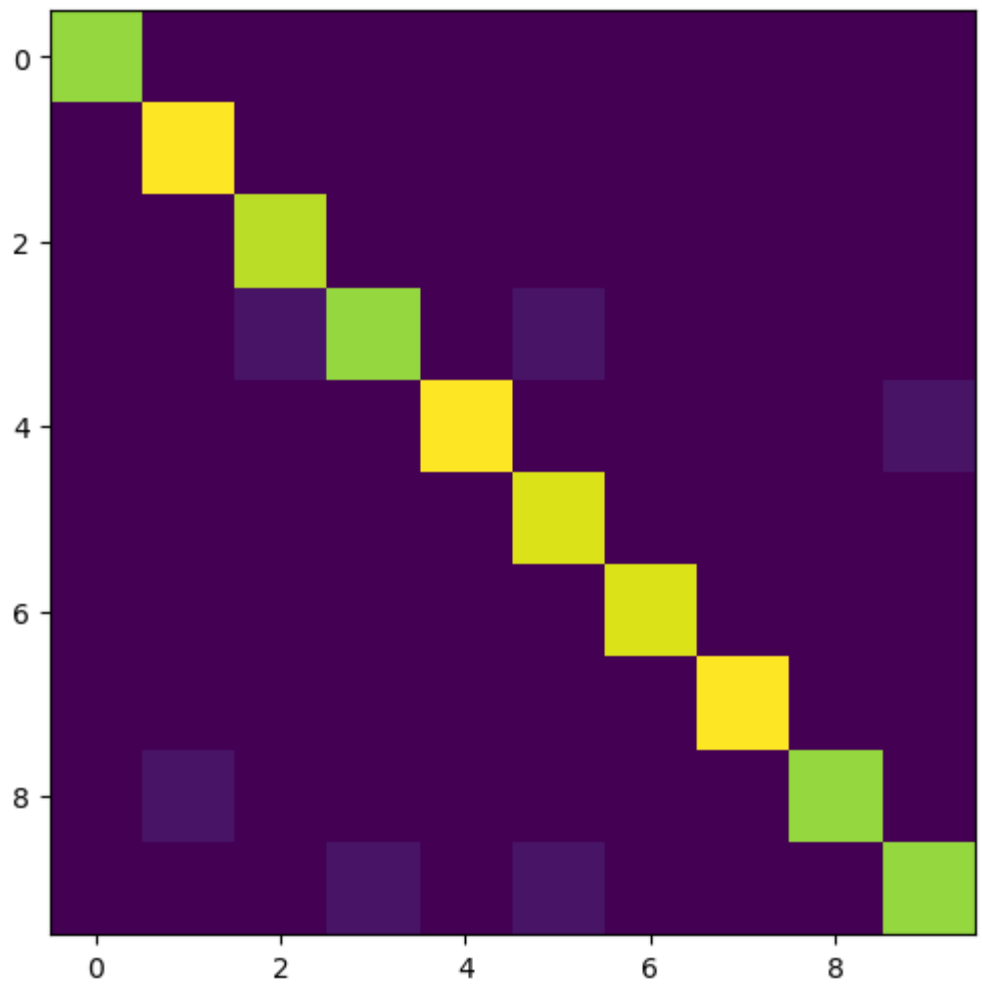
```
4it [00:00, 2343.19it/s]
```




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

In [52]:
```
# TEST/PLOT CODE: DO NOT CHANGE
# TEST kNNClassifier

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

```
4it [00:00, 150.56it/s]
```




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

Ζήτημα 1.5: κ-Κοντινότεροι Γείτονες (k-Nearest Neighbors/kNN) [7 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. Μπορείτε να χρησιμοποιήσετε τη συνάρτηση **np.linalg.norm** για να υπολογίσετε την απόσταση. Το k αναφέρεται στον αριθμό των γειτόνων που συμμετέχουν στην ψηφοφορία για την ομάδα/κλάση.

In [53]:
```python
class kNNClassifier_v1_5():
    def __init__(self, k=3):
        self.k=k

    def train(self, trainData, trainLabels):
        self.X_train = trainData
        self.y_train = trainLabels

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

        Inputs:
        X: Test images (N,64)

        Returns:
        predicted labels (N,)
        """
        """ ==========
        YOUR CODE HERE
        ========== """
        distances = [] #list to store the distances
        for x_test in X: #loop over every test sample
            #calculate the Euclidean distance (L2 norm) between test image and the training samples
            dists = np.linalg.norm(self.X_train - x_test, axis=1)
            distances.append(dists) #append to the list

        distances = np.array(distances) #convert to np array
        indices = np.argsort(distances, axis=1) #indices of the nearest neighbors sorted on ascending

        # Perform voting based on the nearest neighbors
        predicted_labels = [] #list for predicted labels
        for i in range(X.shape[0]): #iterate over test samples
            #get the labels of the k nearest neighbors
            k_nearest_labels = self.y_train[indices[i, :self.k]]
            #get the unique neighboring labels
            unique_labels, label_counts = np.unique(k_nearest_labels, return_counts=True)
            #predict the label by finding the label with the highest count
            predicted_label = unique_labels[np.argmax(label_counts)]
            predicted_labels.append(predicted_label) #append them to list

        return np.array(predicted_labels) #return the array
```

In [54]:
```python
# TEST/PLOT CODE: DO NOT CHANGE
# TEST kNNClassifierManual

knnClassifierManualX = kNNClassifier_v1_5()
knnClassifierManualX.train(X_train, y_train)
print ('kNN classifier accuracy: %f'%test(X_test, y_test, knnClassifierManualX))
```

```
kNN classifier accuracy: 96.666667
```

Ζήτημα 1.6: PCA + κ-κοντινότεροι γείτονες (PCA/k-NN) [8 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 την προηγούμενη υλοποίηση του ταξινομητή kNN σε αυτό το ζήτημα.

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 [55]:

```python
def svd(A):
    """ ==========
    YOUR CODE HERE
    ========== """
    #get the values of svd(a) to the left singular, singular and right transpose
    U, singular_values, Vt = np.linalg.svd(A, full_matrices=False)
    return U, singular_values, Vt.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
        """
        """ ==========
        YOUR CODE HERE
        ========== """
        self.components = components #initialize components
        self.k = k #initialize neighbors involded in voting
        #make an instance of kNNClassifier_v1_5 with k number of neighbors
        self.knn_classifier = kNNClassifier_v1_5(k=k)

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

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

        """ ==========
        YOUR CODE HERE
        ========== """
        self.mean = np.mean(trainData, axis=0) #calculate mean of training images
        #center the training images by subtracting the mean from each image
        X_centered = trainData - self.mean

        """ ==========
        YOUR CODE HERE
        ========== """
        #perform SVD on centered data (mean-deviation form of data matrix)
        U, D, Vt = svd(X_centered)

        #get the principal components of the left singular vector
        self.U = U[:, :self.components]
        #get the principal components of the singular vector
        self.D = D[:self.components]
        #get the principal components of the right transposed singular vector
        self.Vt = Vt[:self.components, :]

        #project the training data onto the principal components
        self.projected_data = np.dot(X_centered, self.Vt.T)

        #store the training labels
        self.trainLabels = trainLabels

        #train k-NN classifier with the projected data and labels
        self.knn_classifier.train(self.projected_data, self.trainLabels)


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

        Inputs:
        x: Test images (N,64)

        Returns:
        predicted labels (N,)
        """
        """ ==========
        YOUR CODE HERE
        ========== """
        #center again the test images
        x_centered = x - self.mean

        #project again test images onto principal components of the right transposed singular vector
        projected_data = np.dot(x_centered, self.Vt.T)

        #predict labels using the k-NN classifier
        predicted_labels = self.knn_classifier(projected_data)
        #return
        return predicted_labels
```

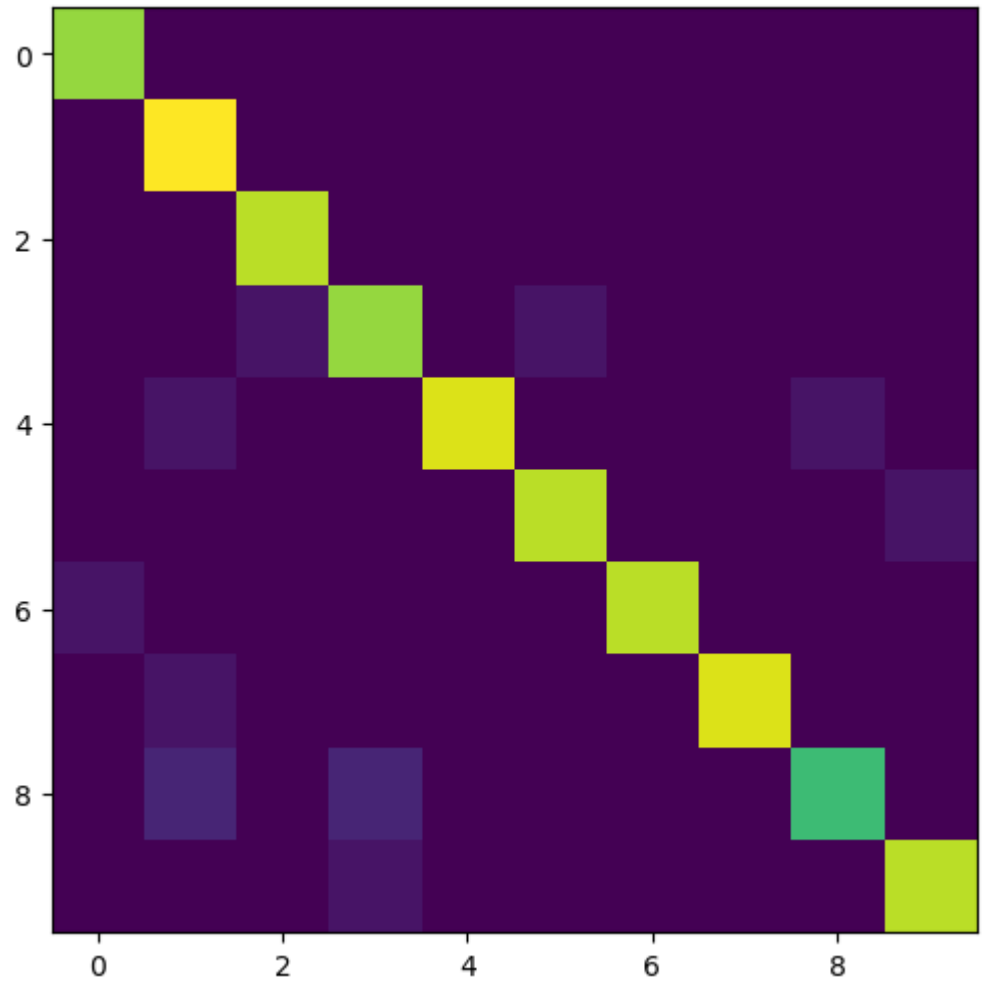
```
88
89
90 # test your classifier with only the first 100 training examples (use this
91 # while debugging)
92 pcaknnClassiferX = PCAKNNClassifer()
93 pcaknnClassiferX.train(X_train[:100], y_train[:100])
94 print ('PCA-kNN classifier accuracy: %f'%test(X_test, y_test, pcaknnClassiferX))
```

```
PCA-kNN classifier accuracy: 80.000000
```

In [56]:

```
 1 # test your classifier with all the training examples
 2 pcaknnClassifer = PCAKNNClassifer()
 3 pcaknnClassifer.train(X_train, y_train)
 4 # display confusion matrix for your PCA KNN classifier with all the training examples
 5 """ ==========
 6 YOUR CODE HERE
 7 ========== """
 8 #confussion matrix and accuracy of pcaknnClassifer
 9 M_pca, acc = Confusion(X_test, y_test, pcaknnClassifer)
10
11 # Display the accuracy and visualize the confusion matrix
12 print ('PCA-kNN classifier accuracy: %f'%test(X_test, y_test, pcaknnClassifer))
13 VisualizeConfussion(M_pca)
```

```
4it [00:00, 72.25it/s]

PCA-kNN classifier accuracy: 93.333333
```




```
[[16.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0. 19.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0. 17.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  1. 16.  0.  1.  0.  0.  0.  0.]
 [ 0.  1.  0.  0. 18.  0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  0. 17.  0.  0.  0.  1.]
 [ 1.  0.  0.  0.  0.  0. 17.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.  0. 18.  0.  0.]
 [ 0.  2.  0.  2.  0.  0.  0.  0. 13.  0.]
 [ 0.  0.  0.  1.  0.  0.  0.  0.  0. 17.]]
```

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

""" WRITE YOUR ANWSER HERE """

Witn kNN we have almost half the time than PCA-kNN that is because in the PCA-kNN we have the training data first projected onto the principal components obtained through singular value decomposition (SVD). This projection step uses matrix multiplications which are computationally expensive, especially when the number of principal components is high. Even though after the projection we have less data and the kNN classification is faster, the multiplications in the begining outweight the faster kNN classification.

Άσκηση 2: Βαθιά Μάθηση [25 μονάδες]

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

- **Τοπικά** (jupyter): Ακολουθήστε τις οδηγίες στη διεύθυνση https://pytorch.org/get-started/locally/ (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 askisi3` , `(base)$ conda activate askisi3` , η εντολή `(askisi3)$ 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 'askisi3', πριν ανοίξετε το jupyter: `(askisi3)$ conda install matplotlib tqdm scipy` και `(askisi3)$ 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: `(askisi3)$ conda install jupyter` και μετά να εκτελέσετε `(askisi3)$ 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 3ης σειράς ασκήσεων, μέσα στο εικονικό περιβάλλον *askisi3* 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. Ο Διδάσκων δεν θα παρέχει καμία υποστήριξη που σχετίζεται με GPU ή την CUDA.

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

In [57]:
```
import scipy
import torch.nn as nn
import torch.nn.functional as F
import torch
from torch.autograd import Variable

# create a tensor of 4x3 random-valued array
x = torch.rand(4, 3)
print(x)
```

```
tensor([[0.6771, 0.7081, 0.0455],
        [0.1154, 0.9941, 0.5876],
        [0.7290, 0.4755, 0.0780],
        [0.7880, 0.8520, 0.4098]])
```

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).

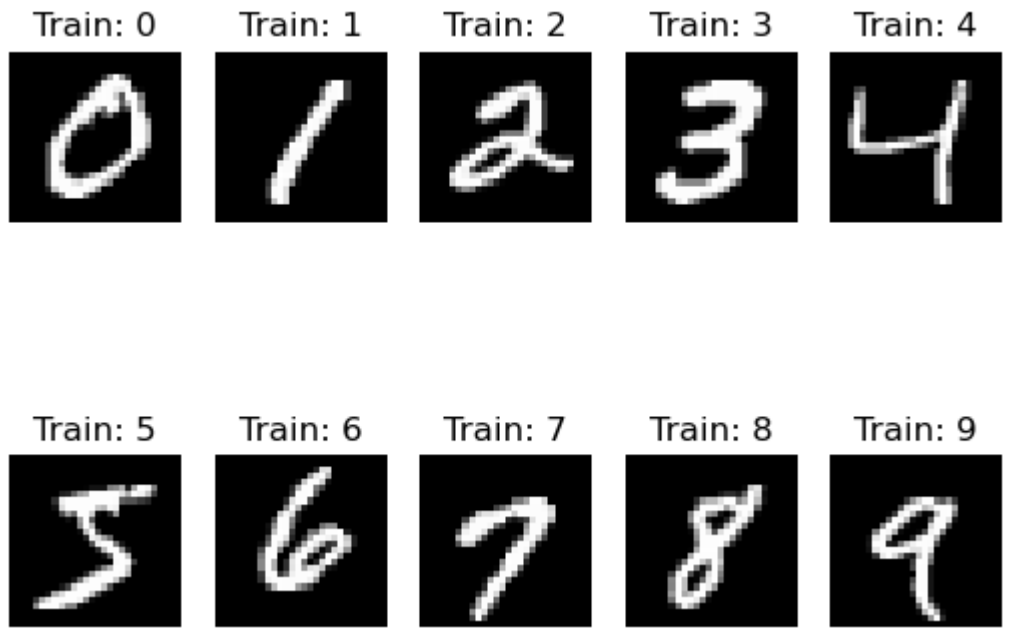
Ο κώδικας που ακολουθεί "κατεβάζει" το σύνολο δεδομένων MNIST της κλάσης torchvision.datasets (https://pytorch.org/vision/main/generated/torchvision.datasets.MNIST.html#torchvision.datasets.MNIST), στο φάκελο `mnist` (του root καταλόγου). Μπορείτε να αλλάξετε τον κατάλογο που δείχνει η μεταβλητή `path` στη διαδρομή που επιθυμείτε. Ενδεικτικό `path` σε περιβάλλον Windows: **path = 'C:/Users/user/Υπολογιστική Όραση/assignments/assignment 3/'**. 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/assignment3/'**.

- 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 [58]:
```
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 [59]:
```
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.

    """
    plt.figure()

    """ ==========
    YOUR CODE HERE
    ========== """

    for i in range(10): #ten images for numbers 0 thought 9
        plot = plt.subplot(2, 5, i + 1) #create a plot with size 2x5 and put it in position i+1
        #display the image of every digit
        #the training dataset X_train is filtered by the target label to equal to i
        #take the first image of the dataset of each digit, we could take any image
        #show the image in grayscale
        plot.imshow(X_train[y_train == i][0], cmap='gray')
        plot.set_title(f'Train: {i}') #set title to Train: i
        plot.axis('off') #remove axis

```

In [60]:
```
# 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 [6 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 (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)` στη σελίδα ecourse του μαθήματος.

In [61]:

```
# 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):
        #choose the loss function, we use CrossEntropyLoss for multi-class classification
        #combination of a softmax operation + log likelihood maximization
        loss_function = nn.CrossEntropyLoss()
        #choose the optimizer using stochastic gradient descent with
        #learning rate is set to 0.01, and the momentum is set to 0.9
        optimizer = optim.SGD(self.parameters(), lr=0.01, momentum=0.9)
        #loop epochs
        for epoch in range(epochs):
            running_loss = 0.0 #set running loss to zero
            #use bar
            pbar = tqdm(range(0, len(X_train), batchSize))
            #loop from 0 to len(X_train) incrementing by batchSize
            for i in pbar:
                #get inputs from each iteration we use FloatTensor to ensure
                #compatability with the neural network
                inputs = Variable(torch.FloatTensor(X_train[i:i+batchSize]))

                #get labels from each iteration we use LongTensor to ensure
                #compatability with the neural network
                labels = Variable(torch.LongTensor(y_train[i:i+batchSize]))

                #resets the gradients of the neural network parameters to zero.
                #It is necessary before performing the backward propagation
                optimizer.zero_grad()  #initialize the gradients to zero

                #forward propagation
                outputs = self.forward(inputs)

                #calculate the estimation error
                loss = loss_function(outputs, labels)

                #backward propagation, autograd magic, computes all the partial derivatives
                loss.backward()

                #update the parameters, take a step in gradient direction
                optimizer.step()

                #update loss by adding the batch loss
                running_loss += loss.item()

                #set bar description
                #show the current epoch number(+1 because we start from 0)
                #and the total number of epochs
                #show the average loss per sample for the current batch
                #and format to 4 decimal digits
                pbar.set_description(f'Epoch {epoch+1}/{epochs} | Loss: {running_loss / (i+batchSize):.4



    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)
```

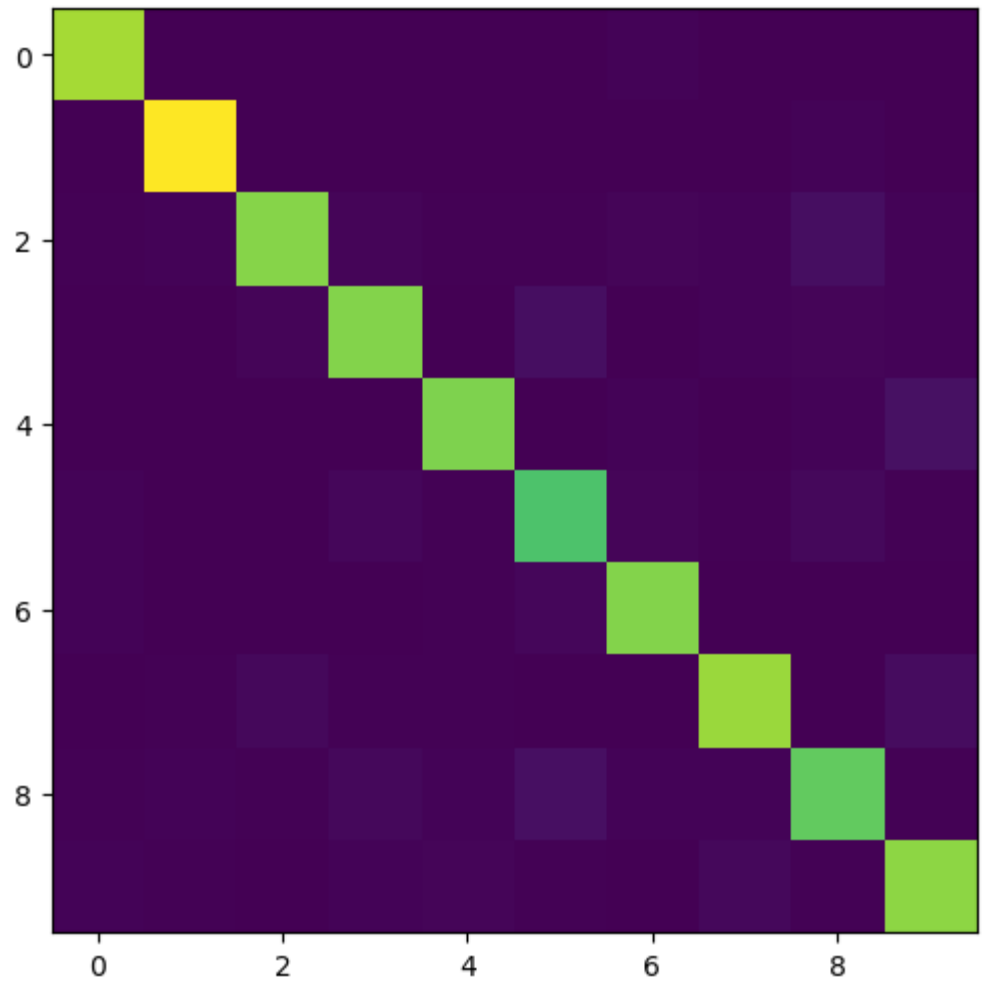
In [62]:
```
# example linear classifier - input connected to output
# you can take this as an example to learn how to extend DNN class
class LinearClassifier(DNN):
    def __init__(self, in_features=28*28, classes=10):
        super(LinearClassifier, self).__init__()
        # in_features=28*28
        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 [63]:
```
# 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: 0.0092: 100%|██████████| 1200/1200 [00:03<00:00, 341.99it/s]
Epoch 2/10 | Loss: 0.0066: 100%|██████████| 1200/1200 [00:03<00:00, 348.10it/s]
Epoch 3/10 | Loss: 0.0062: 100%|██████████| 1200/1200 [00:04<00:00, 297.36it/s]
Epoch 4/10 | Loss: 0.0060: 100%|██████████| 1200/1200 [00:03<00:00, 363.82it/s]
Epoch 5/10 | Loss: 0.0058: 100%|██████████| 1200/1200 [00:03<00:00, 345.28it/s]
Epoch 6/10 | Loss: 0.0057: 100%|██████████| 1200/1200 [00:03<00:00, 359.24it/s]
Epoch 7/10 | Loss: 0.0057: 100%|██████████| 1200/1200 [00:03<00:00, 352.93it/s]
Epoch 8/10 | Loss: 0.0056: 100%|██████████| 1200/1200 [00:03<00:00, 345.43it/s]
Epoch 9/10 | Loss: 0.0055: 100%|██████████| 1200/1200 [00:03<00:00, 364.71it/s]
Epoch 10/10 | Loss: 0.0055: 100%|██████████| 1200/1200 [00:03<00:00, 352.96it/s]

Linear classifier accuracy: 92.090000
```

In [64]:
```
# display confusion matrix
""" ==========
YOUR CODE HERE
========== """
#confussion matrix and accuracy of linearClassifier
M_DNN, accuracy = Confusion(X_test, y_test, linearClassifier)
#round DNN to get integers
rounded_M_DNN = np.round(M_DNN).astype(int)
#display the accuracy and visualize the confusion matrix
VisualizeConfussion(rounded_M_DNN)
```

```
100%|██████████████████████████████████████| 200/200 [00:00<00:00, 4018.13it/s]
```



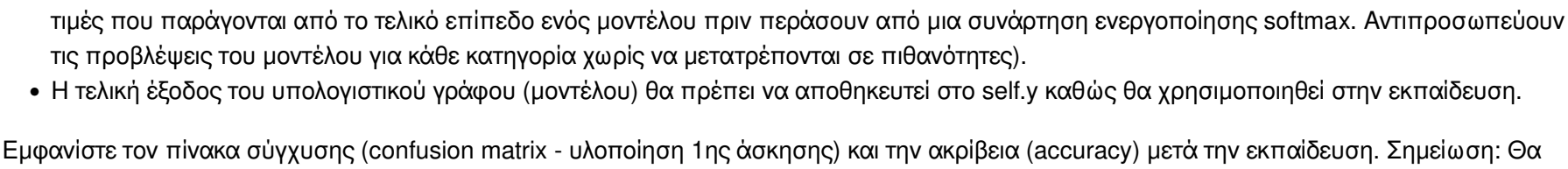

```
[[ 964    0    0    1    0    4    9    1    1    0]
 [   0 1112    2    2    0    1    4    2   12    0]
 [   8   10  911   17    6    5   14   11   40   10]
 [   4    1   17  904    0   43    3   12   15   11]
 [   1    2    3    1  898    1   13    3   10   50]
 [  10    2    2   21    8  800   14    5   23    7]
 [  13    3    3    2    8   18  907    3    1    0]
 [   1    6   22    8    6    1    0  944    3   37]
 [   8   11    6   23    9   45    9   11  844    8]
 [  12    8    1    9   17    8    0   24    5  925]]
```

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 ($z_i = (w_i - min(w))/(max(w) - min(w))$). 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 [65]:
```
# Plot filter weights corresponding to each class, you may have to reshape them to make sense out of the
# linearClassifier.weight1.data will give you the first layer weights
""" ==========
YOUR CODE HERE
========== """
#get the first layer weights
weights = linearClassifier.weight1.data

#normalize the weights to be between 0 and 1
#get smallest weights tensor and subtrack them from weights to get minimum 0 values
#get largest weights tensor and subtrack from them the minimum to get the range of the weights
#divide them to normalize the weights within the range of 0 to 1
weights_normalized = (weights - torch.min(weights)) / (torch.max(weights) - torch.min(weights))

#reshape the weights with dimensions 28x28 and 10 output categories
weights_reshaped = weights_normalized.view(10, 28, 28)

#plot the filter weights using cmap inferno for colored map
fig, axes = plt.subplots(1, 10, figsize=(12, 6))
for i in range(10):
    ax = axes[i]
    ax.imshow(weights_reshaped[i], cmap='inferno')
    ax.set_title(f'Class {i}')
    ax.axis('off')

plt.show()
```



Σχολιασμός των βαρών

Όπως βλέπουμε απο τον colored map τα βαρη παίρνουν τα χαρακτηριστηκα των αντιστοιχων αριθμων που αντιπροσωπευουν. Αυτο γίνεται διοτι το μοντελο συσχετιζει τα μοτιβα των αριθμών απο τις εικονες με τους αριθμούς των κλάσεων. Φαίνεται οτι η κλάση 0 θυμίζει το μηδεν, η 1 εχει την κάθετη γραμμη του αριθμου 1, στην κλαση του 2 βλεπουμε οτι εχει μια κυκλικη μορφη με μια γραμμη στην μεση την οποια δεν αναγνωριζει, στην κλαση του 3 βλεπουμε τρεις διαγωνιες γραμμες που δεν αναγνωριζονται στη μεση και το υπολοιπο σχηματιζει το σχημα του 3 με καμπυλες, στην κλαση του 4 βλεπουμε οριζοντιες γραμμες και μερικες καθετες που θυμιζουν τον αριθμο 4, αντιστοιχα στους υπολοιπους αριθμους με καποιους συνδιασμους καμπυλων και γραμμων παρομοιαζονται και οι λοιποι αριθμοι. Αυτο επιτυγχανεται διοτι το μοντελο οσο μαθαινει προσπαθει να μειωσει το loss function βασισμενο στα training data και στα labels δινοντας μεγαλυτερα βαρη στα pixels με τα που θυμιζουν την καθε κατηγορια αριθμων και καταληγει ετσι σε καποια σχηματα με τα χαρακτηριστηκα των αριθμων.

Ζήτημα 2.4: Νευρωνικό δίκτυο πολλαπλών επιπέδων - Multi Layer Perceptron (MLP) [7 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.

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

Σχεδιάστε 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.

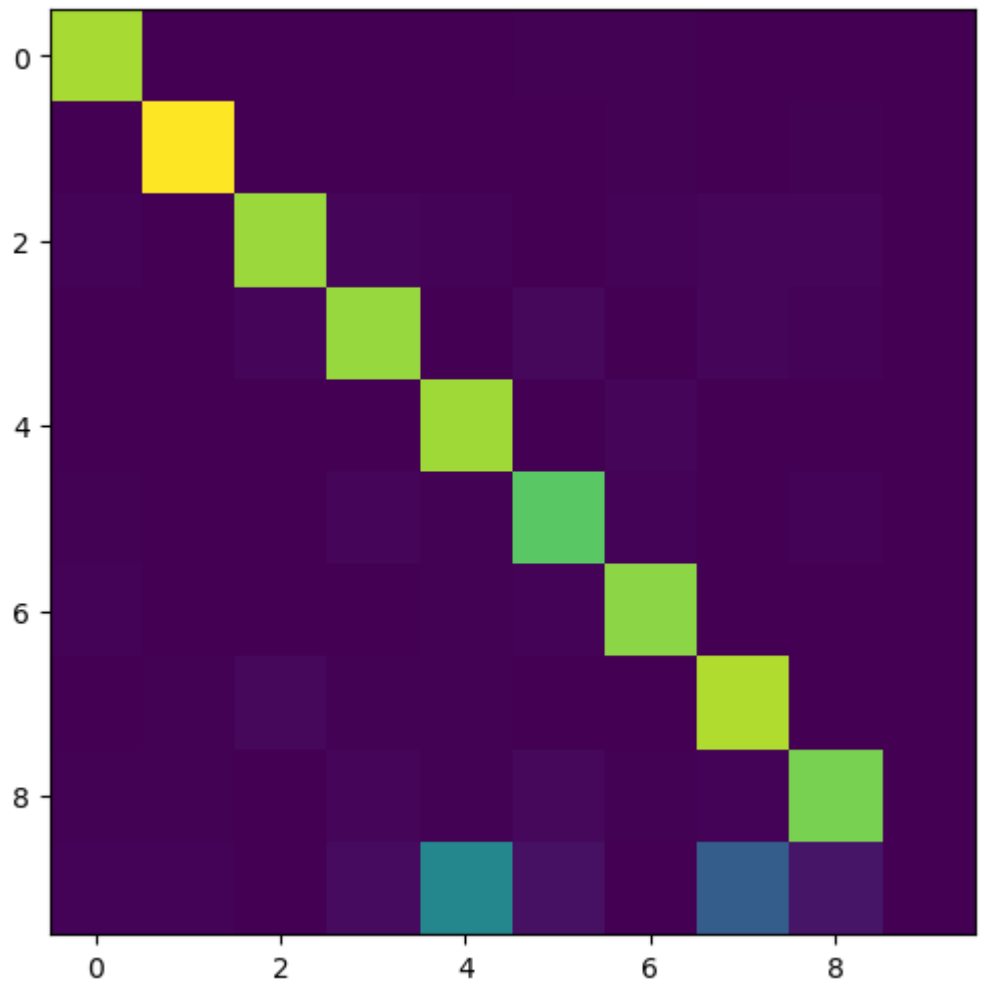
In [66]:

```python
class MLPClassifer(DNN):
    def __init__(self, in_features=28*28, classes=10, hidden=50):
        """
        Initialize weight and bias variables
        """
        super(MLPClassifer, self).__init__()
        """ ==========
        YOUR CODE HERE
        ========== """
        #first linear layer with input the in features
        #and output the hidden layer
        self.fc1 = nn.Linear(in_features, hidden)
        #second linear layer with input the hidden layer
        #and output the classes
        self.fc2 = nn.Linear(hidden, classes)


    def forward(self, x):
        """ ==========
        YOUR CODE HERE
        ========== """
        x = x.view(x.size(0), -1) #reshape x to two dimension tensor
        x = F.relu(self.fc1(x)) #pass the first layer to the Relu
        self.y = self.fc2(x) #pass the second layer without Relu
        self.y = F.softmax(self.y, dim=1) #pass second layer through softmax
        return self.y


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

```
Epoch 1/10 | Loss: 0.0374: 100%|██████████| 1200/1200 [00:04<00:00, 283.18it/s]
Epoch 2/10 | Loss: 0.0331: 100%|██████████| 1200/1200 [00:04<00:00, 299.24it/s]
Epoch 3/10 | Loss: 0.0327: 100%|██████████| 1200/1200 [00:03<00:00, 302.24it/s]
Epoch 4/10 | Loss: 0.0326: 100%|██████████| 1200/1200 [00:04<00:00, 284.20it/s]
Epoch 5/10 | Loss: 0.0325: 100%|██████████| 1200/1200 [00:04<00:00, 284.00it/s]
Epoch 6/10 | Loss: 0.0324: 100%|██████████| 1200/1200 [00:04<00:00, 279.79it/s]
Epoch 7/10 | Loss: 0.0323: 100%|██████████| 1200/1200 [00:04<00:00, 284.21it/s]
Epoch 8/10 | Loss: 0.0323: 100%|██████████| 1200/1200 [00:04<00:00, 280.49it/s]
Epoch 9/10 | Loss: 0.0322: 100%|██████████| 1200/1200 [00:04<00:00, 279.79it/s]
Epoch 10/10 | Loss: 0.0322: 100%|██████████| 1200/1200 [00:04<00:00, 270.56it/s]
```

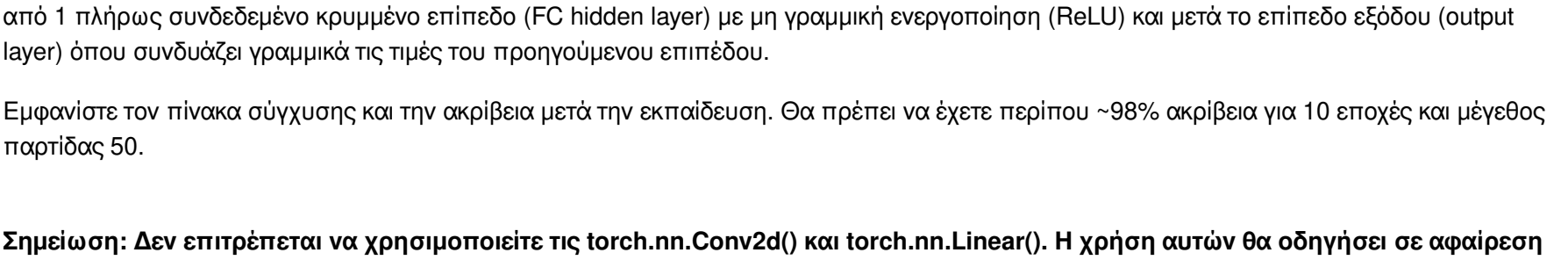
In [67]:
```
# Plot confusion matrix
M_mlp,acc_mlp = Confusion(X_test, y_test, mlpClassifer)

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(np.round(M_mlp).astype(int))
```

```
100%|██████████████████████████████████████| 200/200 [00:00<00:00, 1923.50it/s]

Confusion matrix - MLP classifier accuracy: 85.350000
MLP classifier accuracy: 85.350000
```




```
[[ 966    0    0    0    0    5    6    1    2    0]
 [   0 1112    4    2    1    2    5    2    7    0]
 [  12    2  944   17   12    4   11   14   16    0]
 [   2    0   17  941    1   22    2   14   11    0]
 [   1    1    3    1  955    0   16    2    3    0]
 [   8    2    2   20    6  829   13    2   10    0]
 [  12    3    2    2    5   13  918    2    1    0]
 [   2    7   22    5    6    3    0  980    3    0]
 [   6    5    3   20    6   25    7   12  890    0]
 [  10   11    0   31  514   52    1  328   62    0]]
```

In [68]:
```
# Plot filter weights
""" ==========
YOUR CODE HERE
========== """

#get the first layer weights
weights = mlpClassifer.fc1.weight.data

#normalize the weights to be between 0 and 1
#get smallest weights tensor and subtrack them from weights to get minimum 0 values
#get largest weights tensor and subtrack from them the minimum to get the range of the weights
#divide them to normalize the weights within the range of 0 to 1
normalized_weights = (weights - weights.min()) / (weights.max() - weights.min())
#reshape the weights with dimensions 28x28 and 10 output categories
weights_reshaped = weights_normalized.view(10, 28, 28)

#plot the filter weights using cmap inferno for colored map
fig, axes = plt.subplots(1, 10, figsize=(12, 6))
for i in range(10):
    ax = axes[i]
    ax.imshow(weights_reshaped[i], cmap='inferno')
    ax.set_title(f'Class {i}')
    ax.axis('off')

plt.show()
```



Μοιάζουν τα βάρη αυτά καθόλου με τα βάρη που απεικονίστηκαν στο προηγούμενο ζήτημα; Γιατί ή γιατί όχι?

Ναι μοιαζουν, ωστοσο τωρα εχουμε μεγαλυτερο accuracy. Μοιαζουν διοτι κανουμε normalization και γινονται και εδω rescale σε τιμες μεταξυ 0 και 1. Ωστοσο η μεθοδος που ακολουθειται πανω με εδω ειναι εντελως διαφορετικη καθως εδω εχουμε το hidden layer το οποιο ακολουθεί μια μη γραμμική συνάρτηση ενεργοποίησης ReLU με το οποιο θα επρεπε να εχουμε higher-level ομοιοτητες οι οποιες ισως να μην εμοιαζαν αν δεν καναμε κανονικοποιηση.

Ζήτημα 2.5: Συνελικτικό Νευρωνικό Δίκτυο - Convolutional Neural Network (CNN) [8 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).

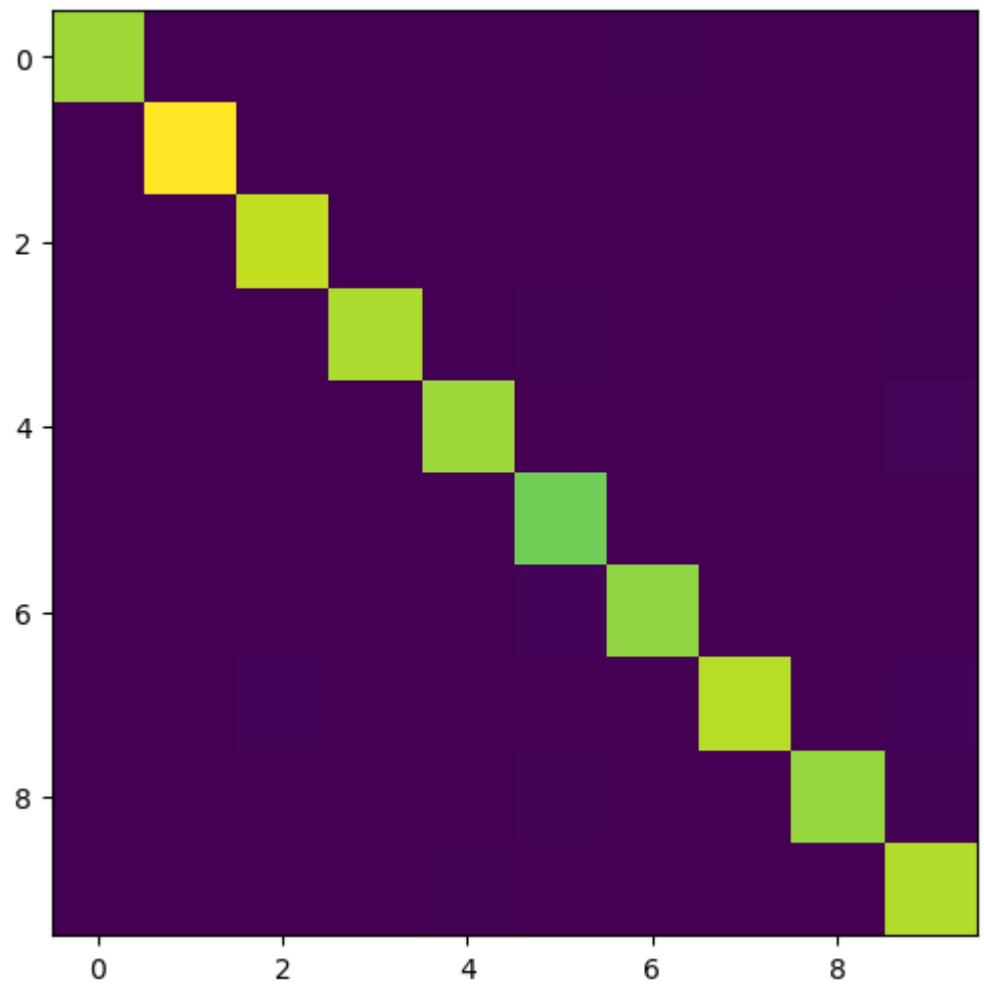
In [69]:

```python
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__()
        """ ==========
        YOUR CODE HERE
        ========== """
        #get weights of the first layer with n the number of output features or filters.
        #The conv1 layer takes an input with grayscale image and applies n filters of size 5x5
        self.conv1_weights = weight_variable((n, 1, 5, 5))
        #the biases for the first convolutional layer with size n
        self.conv1_bias = bias_variable((n))
        #get weights of the seecond layer with 2*n the number of output features or filters.
        #The conv2 layer takes an n input features or filters and applies 2*n filters of size 5x5
        self.conv2_weights = weight_variable((2*n, n, 5, 5))
        #the biases for the second convolutional layer with size 2*n
        self.conv2_bias = bias_variable((2*n))
        #weights of the fully connected hidden layer with 64  hidden units
        #and 7*7*n*2 the size of the flattened output from conv2 7x7 spatial dimensions and n*2 features
        #The linear layer takes the output feature maps from the second convolutional layer
        #which have a spatial dimension of 7x7.
        #This is because the stride of 2 in each convolutional layer
        #reduces the spatial dimensions by a factor of 2.
        #(output_size = (input_size - kernel_size + 2*padding) / stride + 1)
        #after the first conv we have (28 - 5) / 2 + 1 = 12x12
        #and now we get (12 - 5) / 2 + 1 = 7x7
        #The second convolutional layer has 2*n output features or filters.
        #Each of these features produces a 7x7 map.
        #So the number of elements in the spatial maps of the second convolutional layer is 7*7*n*2.
        self.fc_weights = weight_variable((64, 7*7*n*2))
        #the biases for the fully connected hidden layer with size 64
        self.fc_bias = bias_variable((64))
        #weights of the output layer with classes the number of output classes and 64 hidden units
        self.output_weights = weight_variable((classes, 64))
        #the biases for the output layer with size classes
        self.output_bias = bias_variable((classes))

    def forward(self, x):
        """ ==========
        YOUR CODE HERE
        ========== """
        #apply Relu to the first layer weights with stride 2 and bias the conv1_bias
        #and apply it to the tensor x
        x = F.relu(conv2d(x, self.conv1_weights, stride=2, bias=self.conv1_bias))
        #apply Relu to the second layer weights with stride 2 and bias the conv2_bias
        #and apply it to the tensor x
        x = F.relu(conv2d(x, self.conv2_weights, stride=2, bias=self.conv2_bias))
        #reshape x to two dimension tensor with -1 get automaticly the first dimension
        #based on the second one.
        #the second dimension is the feature map after
        #the second convolutional layer with size of 7x7 pixels
        #multiplied with the number of output channels of the second convolutional layer
        #to get the number of elements in each feature map after the second convolutional layer
        x = x.view(-1, 7*7*self.conv2_weights.shape[0])
        #apply Relu to the hidden layer weights multiplied by x and added by the fully connected bias
        #in this way we get a weighted sum of the input features for each neuron in the layer
        #weighted by the corresponding weights, and adjusted by the bias term
        x = F.relu(torch.matmul(x, self.fc_weights.t()) + self.fc_bias)
        #multiply x with the output wights and add the output bias
        #we get raw predictions or logits for each class
        y = torch.matmul(x, self.output_weights.t()) + self.output_bias
        return y

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

```
Epoch 1/10 | Loss: 0.0330: 100%|██████████| 1200/1200 [00:10<00:00, 114.46it/s]
Epoch 2/10 | Loss: 0.0031: 100%|██████████| 1200/1200 [00:10<00:00, 115.08it/s]
Epoch 3/10 | Loss: 0.0017: 100%|██████████| 1200/1200 [00:10<00:00, 113.97it/s]
Epoch 4/10 | Loss: 0.0012: 100%|██████████| 1200/1200 [00:10<00:00, 114.19it/s]
Epoch 5/10 | Loss: 0.0009: 100%|██████████| 1200/1200 [00:10<00:00, 114.09it/s]
Epoch 6/10 | Loss: 0.0007: 100%|██████████| 1200/1200 [00:10<00:00, 113.62it/s]
Epoch 7/10 | Loss: 0.0006: 100%|██████████| 1200/1200 [00:10<00:00, 114.32it/s]
Epoch 8/10 | Loss: 0.0005: 100%|██████████| 1200/1200 [00:10<00:00, 114.76it/s]
Epoch 9/10 | Loss: 0.0004: 100%|██████████| 1200/1200 [00:10<00:00, 115.82it/s]
Epoch 10/10 | Loss: 0.0004: 100%|██████████| 1200/1200 [00:10<00:00, 115.18it/s]
```

In [70]:
```
# Plot confusion matrix and print the test accuracy of the classifier
""" ==========
YOUR CODE HERE
 ========== """
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(np.round(M_cnn).astype(int))
```

```
100%|██████████████████████████████████████| 200/200 [00:00<00:00, 585.48it/s]

Confusion matrix - MLP classifier accuracy: 98.440000
MLP classifier accuracy: 98.440000
```




```
[[ 965    0    2    0    0    0    6    0    4    3]
 [   0 1127    2    1    0    2    1    0    2    0]
 [   2    0 1025    0    1    0    0    1    3    0]
 [   0    0    3  990    1    5    0    4    2    5]
 [   1    0    0    0  963    0    1    0    0   17]
 [   1    0    0    4    0  882    3    0    0    2]
 [   2    2    1    1    1   10  940    0    1    0]
 [   0    3   10    0    0    0    0 1005    1    9]
 [   3    0    3    3    1    6    0    2  950    6]
 [   0    2    0    2    5    2    0    1    0  997]]
```

- 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 στη σελίδα ecourse του μαθήματος **tutorial1_pytorch_introduction.ipynb**.
- Τέλος, μπορείτε ακόμη να πειραματιστείτε (από δικό σας ενδιαφέρον) με ένα demo νευρωνικού δικτύου που δημιουργήθηκε από τους Daniel Smilkov και Shan Carter στη διεύθυνση https://playground.tensorflow.org/ (https://playground.tensorflow.org/) (για πλατφόρμα TensorFlow).

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_3@mye046 onoma.txt assignment3.ipynb assignment3.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 `assignment3.ipynb` στο home directory του Colaboratory (ο κατάλογος home είναι: /content/).
  - Εκτελέστε σε ένα κελί colab ενός νέου notebook: `!jupyter nbconvert --to html /content/assignment3.ipynb`
  - Κάνετε λήψη του assignment3.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.