# Ψηφιακή Επεξεργασία Εικόνας

Αναφορά Β’ Μέρους Εγρασίας

Δροσιάδης Μιχαήλ (ΑΜ: 1066594)

# Πίνακας Περιεχομένων

# Πίνακας Γραφημάτων

# Σκοπός

Επιλέχθηκε το πρώτο θέμα εργασίας: “Κατηγοριοποίηση Εικόνων – Σύγκριση δύο αντιπροσωπευτικών μεθόδων”. Σκοπός της εργασιάς είναι να γίνει κατηγοριοποίηση εικόνων (classification), με χρήση, α) ενός συνελικτικού νευρωνικού δικτύου και β) ενός Support Vector Machine, για το διαχωρισμό των Oriented Gradients των εικόνων.

Οι εικόνες που πρέπει να κατηγοριοποιηθούν είναι εικόνες χειρόγραφων ψηφίων 28x28 pixels, οι οποίες προέρχονται από το dataset MNIST. Οι μέθοδοι έχουν σαν στόχο ένα σύστημα το οποίο θα δέχεται ως είσοδο την εικόνα, και θα δίνει ως έξοδο το ψηφίο που απεικονίζεται σε αυτή, σε μορφή αριθμού από το 0 έως το 9.

# Μορφή δεδομένων

Τα δεδομένα που παρέχει το MNIST dataset έρχονται σε grayscale εικόνες (1 κανάλι των 8bits ανά pixel), με την ανάλογη ετικέτα (label), η οποία παρέχει την πληροφορία του ψηφίου που απεικονίζει η εικόνα. Οι εικόνες είναι χωρισμένες σε δύο διαφορετικά σετ, το σετ εξάσκησης (training set) και το σετ δοκιμής (testing set).

Ενδεικτικές εικόνες του MNIST dataset για κάθε ψηφίο φαίνονται στη παρακάτω εικόνα.

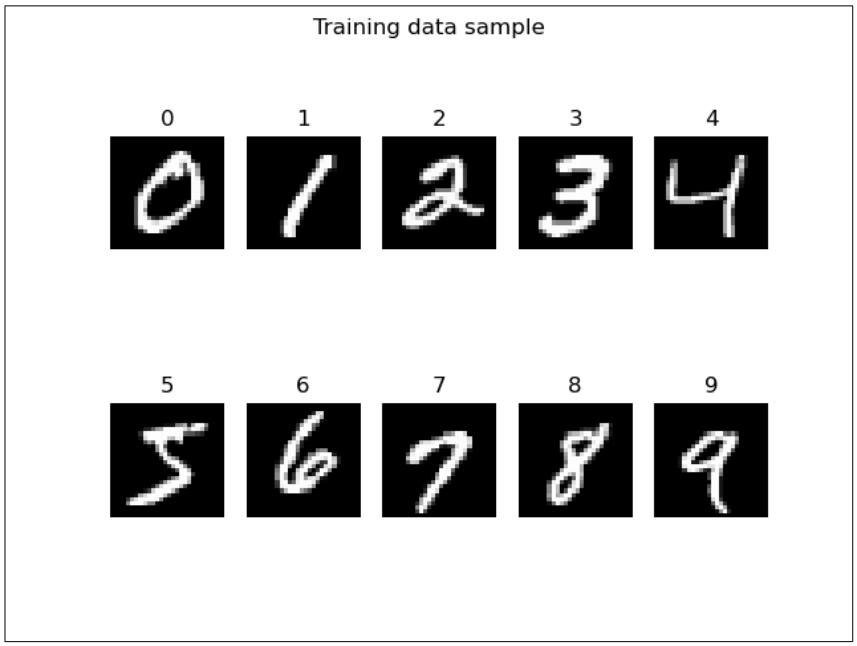


*Figure 1: Ενδεικτικές εικόνες του MNIST dataset για κάθε ψηφίο*

# Μεθοδολογία

Για τη κατηγοριοποίηση των ψηφίων, χρησιμοποιήσαμε δύο μεθόδους: α) ενός συνελικτικού νευρωνικού δικτύου (CNN) και β) ενός Support Vector Machine, για το διαχωρισμό των Oriented Gradients των εικόνων. Και στις δύο περιπτώσεις, η διαδικασία χωρίζεται σε τρία στάδια:

1. Προεπεξεργασία του dataset, ώστε να μπορεί ο εκάστοτε αλγόριθμός να επεξεργαστεί τα δεδομένα
2. Εκπαίδευση του εκάστοτε αλγορίθμου, με το υποσύνολο εκπαίδευσης του dataset
3. Έλεγχος της απόδοσης του εκπαιδευμένου αλγορίθμου, με το υποσύνολο δοκιμής του dataset, άγνωστο ως ώρας για τον αλγόριθμο.

## Μέθοδος με CNN

Ένα τεχνητό νευρωνικό δίκτυο είναι ένα υπολογιστικό σύστημα που εμπνέεται από τη λειτουργία του ανθρώπινου εγκεφάλου. Αποτελείται από διάφορα επίπεδα νευρώνων που συνδέονται μεταξύ τους με τρόπο παρόμοιο με τα βιολογικά νευρωνικά δίκτυα. Κάθε νευρώνας λαμβάνει εισόδους, επεξεργάζεται τις πληροφορίες αυτές και εκπέμπει μια εξόδο.

Τα συνελικτικά νευρωνικά δίκτυα (CNNs) χρησιμοποιούν επίπεδα συνελίξης, που επιτρέπουν στο δίκτυο να ανιχνεύει χαρακτηριστικά σε διάφορα σημεία εικόνων, όπως ακμές, γωνίες και περιοχές με συγκεκριμένα χρώματα.

Για τη σωστή λειτουργία του CNN, οι τιμές των pixels της εικόνας εισόδου πρέπει να μεταφερθούν στο διάστημα 0 έως 1, αντί ακεραίων στο διάστημα 0-255.

Η δομή του CNN που ζητήθηκε περιγράφεται στο παρακάτω σχήμα:

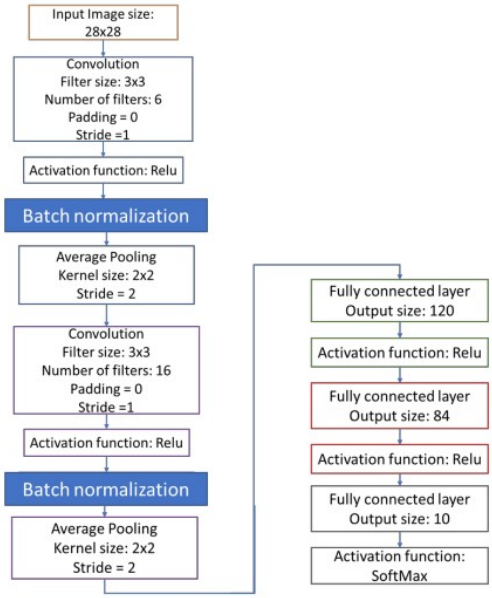


*Figure 2: Δομή του CNN*

Για την υλοποίηση του CNN χρησιμοποιήθηκε η γλώσσα προγραμματισμού Python, με τη βιβλιοθήκη μηχανικής μάθησης PyTorch. Η ίδια παρέχει και τρόπους φόρτωσης και επεξεργασίας του MNIST dataset.

Αφού κατασκευάσαμε τη ζητούμενη δομή και φορτώθηκε το dataset, εκπαιδεύσαμε το CNN για 6 και 12 εποχές, ώστε να μελετήσουμε τα αποτελέσματά του. Για την εκπαίδευση του CNN, μια εποχή είναι ένα “πέρασμα” εκπαίδευσης ολόκληρου του σετ δεδομένων εκπαίδευσης. Με το πέρας κάθε εποχής, αλλά και μετά το πέρας της εκπαίδευσης, ελέγχουμε το CNN με χρήση του σετ δεδομένων δοκιμής. Με αυτά, βγάζουμε το μετρικό ακρίβειας (accuracy), ως το ποσοστό των εισόδων που το CNN κατηγοριοποίησε σωστά.

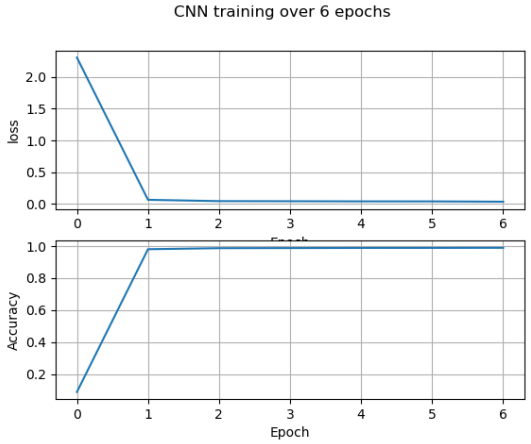


*Figure 3: Μέση τιμή συνάρτησης σφάλματος (loss) και ακρίβεια (accuracy) του CNN σε εκπαίδευση 6 εποχών*

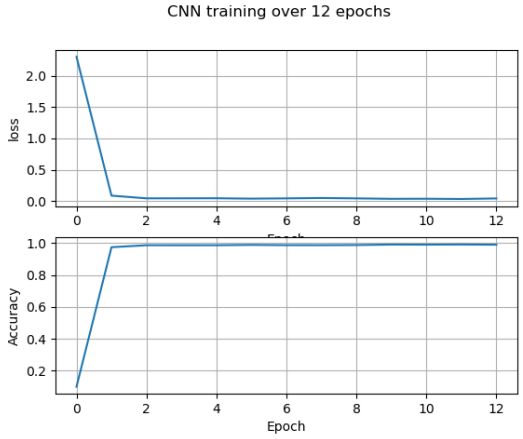


*Figure 4: Μέση τιμή συνάρτησης σφάλματος (loss) και ακρίβεια (accuracy) του CNN σε εκπαίδευση 12 εποχών*

Παρατήρουμε πως το CNN είναι ικανό να φτάσει σε ακρίβεια πάνω του 95% με μόνο ένα πέρασμα των δεδομένων εκπαίδευσης, ενώ, στις μετέπειτα εποχές, το CNN προσπαθεί να ελαχιστοποιήσει περαιταίρω την συνάρτηση σφάλματος, φτάνοντας σε ακρίβεια 99% στις 6 εποχές.

Συγκεκριμένα, η ακρίβεια είναι:

- 9911/10000 (99.11%) στις 6 εποχές (χρόνος εκπαίδευσης 85sec)
- 9896/10000 (98.96%) στις 12 εποχές (χρόνος εκπαίδευσης 170sec)

Η μικρότερη ακρίβεια σε μεγαλύτερο όγκο εκπαίδευσης μπορεί να οφείλεται:

1. Στο ότι το δίκτυο εντόπισε ένα “χειρότερο” τοπικό ελάχιστο στη συνάρτηση σφάλματος, κατά τη βελτιστοποίση με Stochastic Gradient Descent (SGD)
2. Στο ότι ο ρυθμός μάθησησης (learning rate) του SGD είναι σταθερός στο (σχετικά μεγάλο) 0.1. Καθώς το βήμα του SGD παραμένει σταθερό, το δίκτυο μπορεί να εντοπίσει και να

κατεβεί σε ένα ελάχιστο σχετικά γρήγορα (πρώτες εποχές), αλλά έπειτα ταλαντώνεται γύρω από το ελάχιστο, λόγω του μεγάλου βήματος.

Ενδιαφέρον έχει ο υπολογισμός της μήτρας σύγχυσης των αποτελεσμάτων του CNN. Αυτή δείχνει τις προβλέψεις μοντέλου ενάντια στις αναμενόμενες τιμές. Για το δίκτυό μας, αυτές είναι οι εξής:

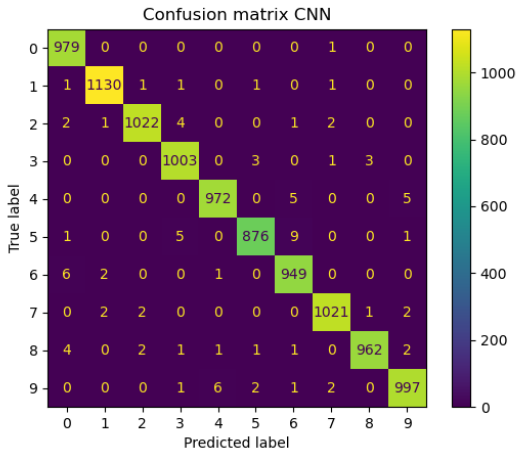


*Figure 5: Μήτρα σύγχυσης στα αποτελέσματα του CNN μετά από 6 εποχές*

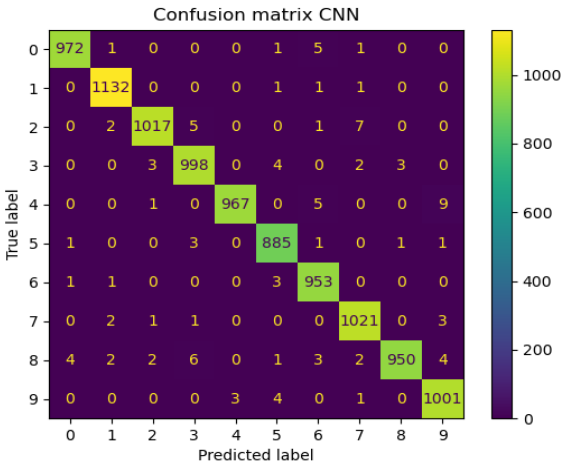


*Figure 6: Μήτρα σύγχυσης στα αποτελέσματα του CNN μετά από 12 εποχές*

## Μέθοδος με HOGs και SVM

Το HOG (Histogram of Oriented Gradients) είναι μια τεχνική εξαγωγής χαρακτηριστικών που χρησιμοποιείται ευρέως στην επεξεργασία εικόνων και την αναγνώριση αντικειμένων. Αυτή η τεχνική συλλέγει πληροφορίες σχετικά με τις κατευθύνσεις και τις έντασεις των ακμών σε μια εικόνα, και στη συνέχεια χρησιμοποιεί αυτές τις πληροφορίες για να περιγράψει την εμφάνιση των αντικειμένων σε αυτήν την εικόνα.

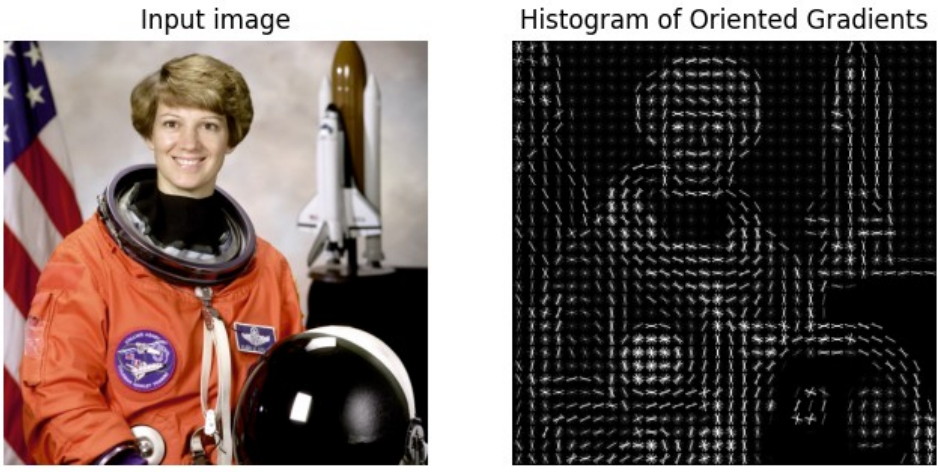


*Figure 7: Παράδειγμα εφαρμογής και οπτικοποίησης των HOGs σε εικόνα*

Μετά τον υπολογισμό των HOGs για την εικόνα, ο οποίος γίνεται σε παράθυρα μέσα στην εικόνα, παράγεται ένα τελικό διάνυσμα, από το συνδιασμό των διανυσμάτων που παράγονται από κάθε ένα από τα (επικαλυπτώμενα) παράθυρα της εικόνας. Αυτό το διάνυσμα ονομάσουμε και HOG descriptor (περιγραφέα).

Ένας τρόπος να κατηγοριοποιήσουμε αυτούς τους περογραφείς είναι με χρήση ενός Support Vector Machine (SVM). Τα SVMs είναι ένα ισχυρό μοντέλο μηχανικής μάθησης που χρησιμοποιείται για την κατηγοριοποίηση και την αναγνώριση προτύπων. Τα SVMs λειτουργούν δημιουργόντας ένα υπερεπίπεδο στον χώρο των χαρακτηριστικών που διαχωρίζει δύο κλάσεις δεδομένων με τέτοιο τρόπο ώστε να μεγιστοποιεί το περιθώριο ανάμεσα στα δύο σύνολα δεδομένων. Τα σημεία δεδομένων που βρίσκονται κοντά στο αυτό το υπερεπίπεδο, τα οποία ονομάζονται διανύσματα υποστήριξης, είναι τα πιο σημαντικά για την απόφαση της κατηγοριοποίησης.

Στις δοκιμές μας, δημιουργήσαμε HOGs, τα οποία μετά κατηγοριοποήσαμε με χρήση SVM με μέγεθος κελιού 4x4, 6x6 και 8x8 pixels. Μετά τη προσαρμογή (fitting) του SVM σε κάθε περίπτωση, χρημοποιώντας τα δεδομένα δοκιμής, η μέθοδος αποδίδει τα παρακάτω αποτελέσματα στα δεδομένα ελέγχου:

- Κελί 4x4: 9891 / 10000 (ακρίβεια: 98.91%, χρόνος υπολογισμού HOGs 86sec, χρόνος fitting 363sec, σύνολο 449sec)
- Κελί 6x6: 9829 / 10000 (ακρίβεια: 98.29%, χρόνος υπολογισμού HOGs 30sec, χρόνος fitting 103sec, σύνολο 134sec)
- Κελί 8x8: 9724 / 10000 (ακρίβεια: 97.24%, χρόνος υπολογισμού HOGs 24sec, χρόνος fitting 68sec, σύνολο 94sec)

Με χρήση μικρότερου κελιού επιτυγχάνουμε καλύτερη ακρίβεια, το μέγεθος του μοντέλου και ο χρόνος εκπαίδευσης αυξάνονται σημαντικά, με το μοντέλο 4x4 να χρειάζεται 106MB(!) για να αποθηκευθεί στη μνήμη.

Οι μήτρες σύγχυσης για κάθε μέγεθος κελιού έχουν ως εξής:

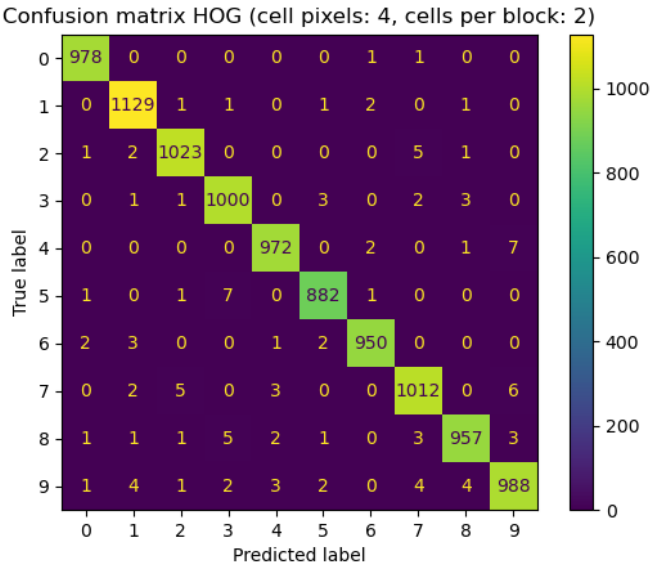


*Figure 8: Μήτρα σύγχυσης της τεχνικής SVM πάνω σε HOGs (μέγεθος κελιού 4x4)*

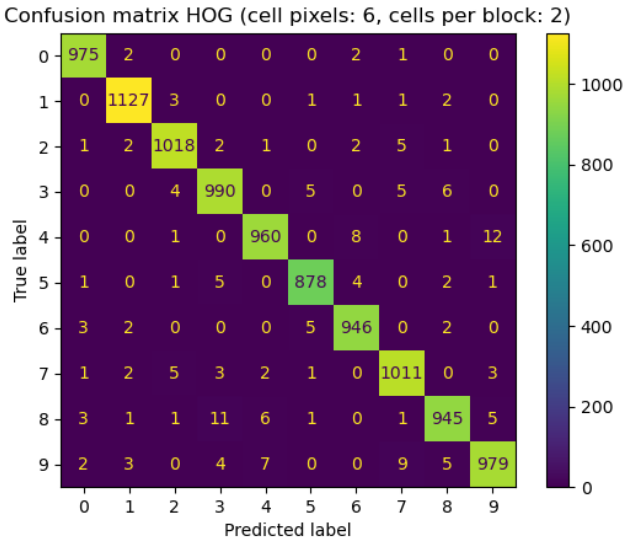


*Figure 9: Μήτρα σύγχυσης της τεχνικής SVM πάνω σε HOGs (μέγεθος κελιού 6x6)*

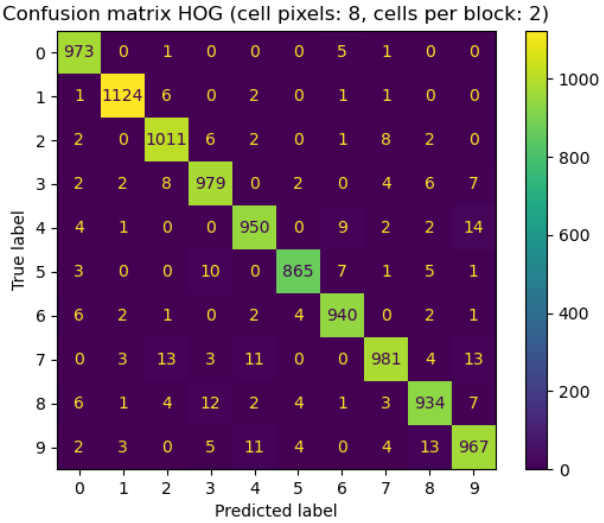


*Figure 10: Μήτρα σύγχυσης της τεχνικής SVM πάνω σε HOGs (μέγεθος κελιού 8x8)*

# Συμπεράσματα

Μετά τον πειραματισμό μας, συμπεραίνουμε πως και οι δύο τεχνικές που δοκιμάσαμε μπορούν να κατηγοριοποιήσουν ικανοποιητικά το dataset χειρόγραφων ψηφίων MNIST. Ωστόσο, πρέπει να σημειώσουμε πως το CNN απέδωσε καλύτερα, με βάση τα αποτελέσματά του και του μακράν γρηγορότερου χρόνου εκπαίδευσης (συμπεριλαμβανομένου ελέγχου μετά από κάθε εποχή!), όπως φαίνεται στο παρακάτω γράφημα:

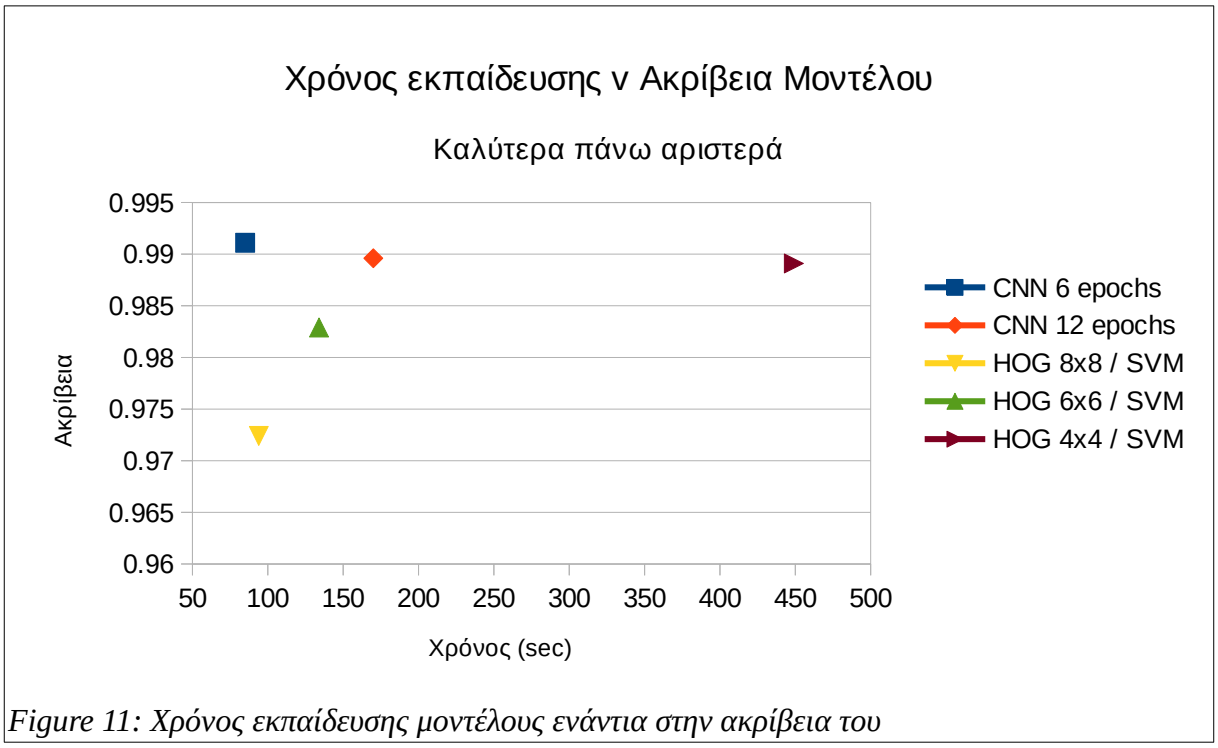


*Figure 11: Χρόνος εκπαίδευσης μοντέλους ενάντια στην ακρίβεια του*

Η ικανότητα των πυρήνων συνέλιξης να εντοπίζουν χαρακτηριστικά σε γειτνιακά pixels φαίνεται ζωτικής σημασίας στο πρόβλημα αναγνώρισης χειρόγραφων ψηφίων.

# Λεπτομέρειες υλοποίησης

Ο κώδικας που αναπτύχθηκε για την εργασία είναι γραμμένος σε Python 3. Λόγω του χρόνου εκπαίδευσης, κάθε ένα από τα μοντέλα αποθηκεύονται για μετέπειτα άμεση χρήση, στα αρχεία ***model_cnn.data*** και ***model_hog.data***. Για τη προσέγγιση των HOGs, αποθηκεύονται ακόμα οι μετασχηματισμοί HOG στο αρχείο ***data/hog_dataset.gz***, καθώς ο υπολογισμός τους είναι χρονοβόρος. Τα δεδομένα του dataset MNIST κατεβάζονται από τη βιβλιοθήκη PyTorch στο

φάκελο ***data/MNIST***. Τα αρχεία ***src/cnn.py*** και ***src/hog.py*** υλοποιούν τις δύο προσεγγίσεις κατηγοροποίησης. Στο αρχείο ***src/utilities.py*** βρίσκονται βοηθητικές συναρτήσεις για τον υπολογισμό της μήτρας σύγχυσης.

Οδηγίες εγκατάστασης και εκτέλεσης βρίσκονται στο αρχείο ***README.md***.

Ο κώδικας της εργασίας βρίσκεται στο github: https://github.com/mdrosiadis/project-psifiaki-epexergasia-eikonas

# Παράρτημα: Κώδικας εργασιάς

## src/cnn.py

```
import os
from lz4.frame import create_compression_context
from matplotlib.cbook import print_cycles
import torch
import torch.nn as nn

from torchvision import datasets
from torchvision.transforms import ToTensor

from torch.utils.data import DataLoader
from torch.autograd import Variable

from torch.optim import SGD

from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt

from utilities import create_confusion_matrix


MODEL_FILENAME = 'model_cnn.data'
BATCH_SIZE = 100
NUM_EPOCHS = 6

# download MNIST dataset
train_data = datasets.MNIST(
    root = 'data',
    train = True,
    transform = ToTensor(),
    download = True,
)
test_data = datasets.MNIST(
    root = 'data',
    train = False,
    transform = ToTensor()
)

loaders = {
    'train' : DataLoader(train_data,
                         batch_size=BATCH_SIZE,
                         shuffle=True,
                         num_workers=1),

    'test'  :DataLoader(test_data,
                        batch_size=BATCH_SIZE,
                        shuffle=True,
                        num_workers=1),
}

# display 1 image from each class
train_targets_list = [t.item() for t in train_data.targets]
fig = plt.figure()
for label in range(10):
    i = train_targets_list.index(label)
    plt.subplot(2, 5, label+1)
    plt.axis('off')
    plt.imshow(train_data.data[i], cmap='gray', vmin=0, vmax=255)
    plt.title(str(label))

plt.gcf().suptitle('Training data sample')
# plt.show()
plt.savefig('plots/dataset_preview.png')

# Device configuration
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
# network class
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(
                in_channels=1,
                out_channels=6,
                kernel_size=3,
                stride=1,
                padding=0,
            ),
            nn.ReLU(),
            nn.BatchNorm2d(6),
            nn.AvgPool2d(kernel_size=2, stride=2),
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(
                in_channels=6,
                out_channels=16,
                kernel_size=3,
                stride=1,
                padding=0,
            ),
            nn.ReLU(),
            nn.BatchNorm2d(16),
            nn.AvgPool2d(kernel_size=2, stride=2),
        )
        self.full1 = nn.Sequential(
            nn.Linear(16*5*5, 120),
            nn.ReLU()
        )
        self.full2 = nn.Sequential(
            nn.Linear(120, 84),
            nn.ReLU()
        )
        self.out = nn.Sequential(
            nn.Linear(84, 10),
            # nn.Softmax(dim=1)
        )


    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(x.size(0), -1)
        x = self.full1(x)
        x = self.full2(x)

        return self.out(x)

    def predict_(self, images):
        test_output = model(images)
        pred_y = torch.max(test_output, 1)[1].data.squeeze()
        return pred_y

    def predict_one(self, img):
        return self.predict_(img.resize(1, 1, 28, 28))



def train(num_epochs, model, loaders):
    loss_func = nn.CrossEntropyLoss()
    optimizer = SGD(model.parameters(), lr=0.1, momentum=0.9)
    model.train()
    # Train the model
    total_step = len(loaders['train'])
    epoch_ticks = [0]
    ac, l = test(model, loss_func=loss_func)
    accuracy_val = [ac]
    loss_val = [l]
    for epoch in range(num_epochs):
        model.train()
        for i, (images, labels) in enumerate(loaders['train']):
```

```
            b_x = Variable(images)   # batch x
            b_y = Variable(labels)   # batch y

            output = model(b_x)
            loss = loss_func(output, b_y)

            # clear gradients for this training step
            optimizer.zero_grad()
            # backpropagation, compute gradients
            loss.backward()               # apply gradients
            optimizer.step()
            if (i+1) % 100 == 0:
                print('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'
                      .format(epoch + 1, num_epochs, i + 1, total_step, loss.item()))

        epoch_ticks.append(epoch+1)
        ac, l = test(model, loss_func=loss_func)
        accuracy_val.append(ac)
        loss_val.append(l)

    plt.subplot(2, 1, 1)
    plt.plot(epoch_ticks, loss_val)
    plt.grid()
    plt.xlabel('Epoch')
    plt.ylabel('loss')

    plt.subplot(2, 1, 2)
    plt.plot(epoch_ticks, accuracy_val)
    plt.grid()
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')

    plt.gcf().suptitle(f'CNN training over {num_epochs} epochs')
    plt.savefig(f'plots/cnn_training_{NUM_EPOCHS}.png')
    # plt.show()

# b_y = Variable(lab)   # batch y
def test(model, loss_func=None,print_output=False):
    # Test the model
    model.eval()
    with torch.no_grad():
        correct = 0
        total = 0
        ls = []
        for images, labels in loaders['test']:
            # test_output = model(images)
            # pred_y = torch.max(test_output, 1)[1].data.squeeze()
            pred_y = model.predict_(images)
            correct += (pred_y == labels).sum().item()
            total += labels.size(0)
            if loss_func is not None:
                ls.append(loss_func(model(Variable(images)), Variable(labels)).item())
            # accuracy = (pred_y == labels).sum().item() / float(labels.size(0))
            # print(pred_y, labels, accuracy)

        avg_loss = None
        if loss_func is not None:
            avg_loss = sum(ls) / len(ls)

        acc = correct / total
        if print_output:
            print('Test Accuracy of the model on the 10000 test images: %.2f' % acc)
            print(f'{correct=} {total=} {len(loaders["test"])=}')


    return acc, avg_loss


model = CNN()
if not os.path.isfile(MODEL_FILENAME):
    print('Training model')
    train(NUM_EPOCHS, model, loaders)
    torch.save(model.state_dict(), MODEL_FILENAME)
else:
    print('Using pretrained model')
```

```
    model.load_state_dict(torch.load(MODEL_FILENAME))
    model.eval()

print('Trained model')
test(model, print_output=True)

print("Testing model against test data")
model.eval()
with torch.no_grad():
    actual = []
    predictions = []
    for images, labels in loaders['test']:
        # test_output = model(images)
        # pred_y = torch.max(test_output, 1)[1].data.squeeze()
        pred_y = model.predict_(images)
        actual.extend([l.item() for l in labels])
        predictions.extend([p.item() for p in pred_y])


cm = create_confusion_matrix(actual, predictions)
print('Confusion matrix')
print(cm)
# plot confusion matrix
ConfusionMatrixDisplay.from_predictions(actual, predictions)
plt.title('Confusion matrix CNN')
plt.savefig(f'plots/confusion_cnn_{NUM_EPOCHS}.png')
# plt.show()
```

# src/hog.py

```
import matplotlib.pyplot as plt
from skimage.feature import hog
from torchvision import datasets
from torchvision.transforms import ToTensor
from sklearn import svm
from joblib import dump, load
import os

from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt

from utilities import create_confusion_matrix

DATASET_FILENAME = 'data/hog_dataset.gz'
MODEL_FILENAME = 'model_hog.data'

CELL_PIXELS = 8
CELLS_PER_BLOCK = 2

if not os.path.isfile(DATASET_FILENAME):
    print('Computing HOGs for MNIST dataset')
    # create the dataset
    # download MNIST dataset
    train_data = datasets.MNIST(
        root = 'data',
        train = True,
        transform = ToTensor(),
        download = True,
    )
    test_data = datasets.MNIST(
        root = 'data',
        train = False,
        transform = ToTensor()
    )

    # transform images to HOGs
    train_hogs = [
        hog(image, orientations=9, pixels_per_cell=(CELL_PIXELS, CELL_PIXELS), cells_per_block=(CELLS_PER_BLOCK,
CELLS_PER_BLOCK))
        for image in train_data.data
    ]
    train_targets = train_data.targets
```

```
    test_hogs = [
        hog(image, orientations=9, pixels_per_cell=(CELL_PIXELS, CELL_PIXELS), cells_per_block=(CELLS_PER_BLOCK,
CELLS_PER_BLOCK))
        for image in test_data.data
    ]
    test_targets = test_data.targets

    hog_dataset = (train_hogs, train_data.targets, test_hogs, test_data.targets)
    # save dataset to file
    dump(hog_dataset, DATASET_FILENAME)

else:
    # load dataset
    print('Loading cached HOGs')
    train_hogs, train_targets, test_hogs, test_targets = load(DATASET_FILENAME)


if not os.path.isfile(MODEL_FILENAME):
    # create the SVM
    print("Fitting SVM")
    clf = svm.SVC(decision_function_shape='ovo') # one versus one (ovo)
    # fit to train data
    clf.fit(train_hogs, train_targets)
    # save trained model to file
    print('Saving fitted SVM')
    dump(clf, MODEL_FILENAME)

else:
    print("Using pretrained SVM model")
    # load the model
    clf = load(MODEL_FILENAME)


# test against testing data
print("Testing model against test data")
test_predictions = clf.predict(test_hogs)
comp = [tp == tt for tp, tt in zip(test_predictions, test_targets)]

correct = sum(comp).item()
total = len(comp)
accuracy = correct / total
print(f'Correct: {correct} / {total} (accuracy: {accuracy})')

cm = create_confusion_matrix(test_targets, test_predictions)
print('Confusion matrix')
print(cm)
# plot confusion matrix

ConfusionMatrixDisplay.from_predictions(test_targets, test_predictions)
plt.title(f'Confusion matrix HOG (cell pixels: {CELL_PIXELS}, cells per block: {CELLS_PER_BLOCK})')
plt.savefig(f'plots/confusion_hog_{CELL_PIXELS}_{CELLS_PER_BLOCK}.png')

# plt.show()
```

# src/utilities.py

```
import numpy as np

def create_confusion_matrix(actual, predictions):
    cm = np.zeros((10, 10), dtype=int)
    for a, p in zip(actual, predictions):
        cm[a][p] += 1

    return cm
```