

# Οπτική Αναγνώριση Χαρακτήρων

Οδυσσέας Σοφικίτης, 10130  
sodyssea@ece.auth.gr

# 1 Εισαγωγή

Η διαδικασία που ακολουθήθηκε είναι ίδια με αυτή που περιγράφεται στην εκφώνηση με κάποια επιπρόσθετα βήματα προεπεξεργασίας σε κάθε στάδιο για βελτίωση της εικόνας εισόδου και καλύτερα αποτελέσματα. Αρκέτα από αυτά τα βήματα δημιουργήθηκαν στην αρχή λόγω της ποιότητας της αρχικά δούθείσας εικόνας που χρειαζόταν ιδιαίτερη μεταχείριση ώστε να ξεχωρίζονται τα γράμματα. Τελικά, η εκπαίδευση έγινε με βάση τις επόμενες εικόνες και κείμενα, αλλά παρ' όλ' αυτά τα βήματα διατηρήθηκαν αφού είχαν καταφέρει να πετύχουν σχετικά καλά αποτελέσματα μέχρι και σε εικόνες σαν την αρχική, οπότε γενικεύουν την ικανότητα του συστήματος που αναπτύχθηκε. Στην γενική περίπτωση οι εικόνες που χρησιμοποιούνται μετατρέπονται σε *binary* και σε σταθερό μέγεθος για το οποίο έχει βελτιστοποιηθεί το σύστημα (ή τα μεγέθη που χρησιμοποιούνται είναι ανάλογα των διαστάσεων).

## 2 Διόρθωση Γωνίας Εικόνας

### 2.1 Εντοπισμός Γωνίας

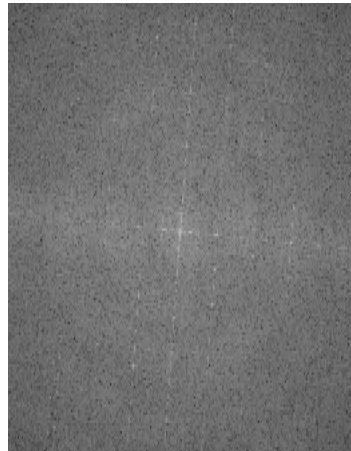
Η λογική είναι πως εφόσον είχαμε μια εικόνα με τέλειες λευκές και μαύρες γραμμές τότε το μέτρο του μετασχηματισμού *Fourier* της εικόνας θα ήταν δύο διαμετρικά τοποθετημένες ως προς το κέντρο της εικόνας *Fourier* (εφόσον έχει γίνει *shifted* ώστε να είναι οι χαμηλές συχνότητες στο κέντρο). Η ευθεία που ενώνει τα σημεία αυτά (ή ένα από τα σημεία και το κέντρο) σχηματίζει με τον κατακόρυφο άξονα γωνία ίδια με την γωνία που σχηματίζει το διάνυσμα των συχνοτήτων που έχει η εικόνα, οπότε αρκεί να βρούμε ένα από αυτά τα σημεία για να αναιρέσουμε την περιστροφή της εικόνας. Η διαδικασία αυτή δεν μπορεί να εφαρμοστεί άμεσα σε μια εικόνα κειμένου, αλλά χρειάζεται πρώτα μια επεξεργασία και μετά ένα *fine-tuning*.

Αρχικά, ”θολώνουμε” την εικόνα (στην συγκεκριμένη υλοποίηση χρησιμοποιείται `imgaussfilt`) για να ενωθούν οι λέξεις μεταξύ τους ώστε να μιμηθούμε μια ενιαία μαύρη γραμμή και στην συνέχεια παίρνουμε τον λογάριθμο του μέτρου του *Fourier*. Στα Σχήματα 2.1 - 2.2 φαίνεται ο μετασχηματισμός *Fourier* χωρίς και με θόλωμα, αντίστοιχα. Όπως βλέπουμε στην δεύτερη περίπτωση έχει αφαιρεθεί η πλειονότητα των συνιστωσών που υπάρχουν στην πρώτη, κάνοντας πιο ευδιάκριτη την ευθεία που χρειάζεται να εντοπίσουμε. Επειδή στην κατεύθυνση της ευθείας περιμένουμε να έχουμε την μέγιστη συχνότητα μεταβολής, ουσιαστικά αναζητούμε το μέγιστο της εικόνας.

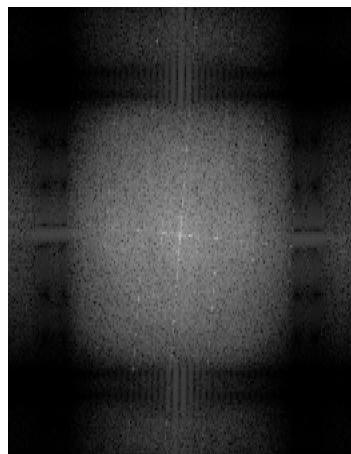
Αν το αναζητήσουμε έτσι όπως είναι η εικόνα, δεν θα βρούμε ένα σημείο της ευθείας αλλά ένα σημείο της *DC* συνιστώσας. Για τον λόγο αυτό, αφαιρούμε τις κεντρικές συχνότητες με μία κυκλική μάσκα και μετά παίρνουμε το μέγιστο (Σχήμα 2.3). Το μέγεθος της μάσκας βρέθηκε κυρίως με *trial and error* ώστε να λειτουργεί καλά για διάφορες εικόνες και να δίνει καλή πρώτη εκτίμηση για την γωνία. Αφού βρούμε το σημείο του μεγίστου, υπολογίζουμε την κλίση της ευθείας με τον κατακόρυφο άξονα και έχουμε την πρώτη γωνία,  $\hat{\theta}$ . Αφού βρούμε την γωνία αυτή, δοκιμάζουμε τις γωνίες  $\theta \in [\hat{\theta} - 4, \hat{\theta} + 4]$ , χρησιμοποιώντας βήμα 0.1 και αναίρωντας την περιστροφή για κάθε μία από αυτές. Ως κριτήριο επιλογής για την καλύτερη γωνία χρησιμοποιείται το άθροισμα των απόλυτων διαφορών της προβολής της εικόνας στον κατακόρυφο άξονα, `sum(abs(diff(proj)))`.

Οι επιλογές αυτές είναι καθαρά σχεδιαστικές και έγιναν με ορισμένες δοκιμές. Καλύπτουν ένα σχετικά ευρύ φάσμα εικόνων και όλες τις γωνίες περιστροφής ( $\theta \in [-90, 90]$ ), αλλά

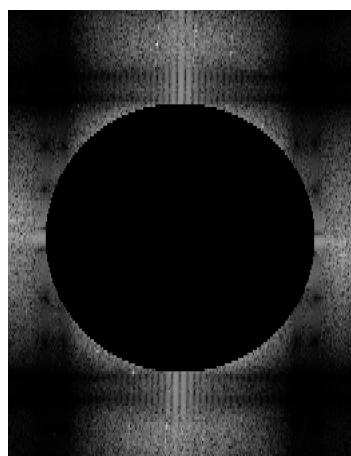
προφανώς θα μπορούσαν να γίνουν διαφορετικά (διαφορετικός συντελεστής θόλωσης, άλλη μάσκα, μικρότερο διάστημα αναζήτησης, μικρότερο βήμα κτλ).



Σχήμα 2.1: Μέτρο μετασχηματισμού *Fourier* χωρίς θόλωμα (zoomed)



Σχήμα 2.2: Μέτρο μετασχηματισμού *Fourier* με θόλωμα (zoomed)



Σχήμα 2.3: Εφαρμογή μάσκας για εύρεση του μεγίστου (zoomed)

## 2.2 Περιστροφή εικόνας

Για την περιστροφή της εικόνας λειτουργούμε περισσότερο με την κλασσική έννοια των συντεταγμένων (όχι σαν να έχουμε έναν δισδιάστατο πίνακα που είναι εικόνα) και εφαρμόζουμε δύο αλλαγές συστήματος συντεταγμένων και μία περιστροφή. Ουσιαστικά, μεταποίζουμε κάθε σημείο ώστε η ύσεη του στον πίνακα να είναι σε συντεταγμένες με αρχή το κέντρο του πίνακα, εφαρμόζουμε τον πίνακα περιστροφής:

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \quad (1)$$

και αλλάζουμε πάλι σύστημα συντεταγμένων ώστε η αρχή να είναι η πάνω αριστερά γωνία της εικόνας.

Η `rotateImage` έχει υλοποιηθεί είτε για *binary* είτε για *RGB* εικόνες, ώστε κατά τον εντοπισμό της γωνίας (που γίνεται πολλές φορές η περιστροφή) να χρησιμοποιείται η *binary* εικόνα επειδή παρατηρήθηκε πως είναι πιο γρήγορο. Η τελική περιστροφή εφαρμόζεται στην *RGB* εικόνα, αλλιώς θα χαλάσει πολύ η ομαλότητα των γραμμάτων, ιδιαίτερα μετά το *upsampling*. Τέλος, θα μπορούσε να βελτιωθεί η ταχύτητα άμα άλλαζε η υλοποίηση με χρήση *vectorization*. Από εδώ και πέρα χρησιμοποιούνται μόνο *binary* εικόνες.

## 3 Εύρεση χαρακτήρων

### 3.1 Διόρθωση εικόνας

Πριν προχωρήσουμε στον εντοπισμό των γραμμάτων, αλλάζουμε το μέγευθος της εικόνας. Διατηρούμε το *aspect ratio*, αλλά την αλλάζουμε έτσι ώστε η μεγαλύτερη διάσταση να είναι *4K*. Αυτό γίνεται γιατί τα *structure elements* που χρησιμοποιούμε βελτιστοποιήθηκαν για εικόνες των συγκεκριμένων διαστάσεων. Ακόμη, αφαιρούμε το μαύρο φόντο που τυχόν υπάρχει λόγο των περιστροφών.

### 3.2 Αρχικός εντοπισμός γραμμάτων

Για τον εντοπισμό των γραμμάτων χρειάζεται πρώτα να εντοπίσουμε τις σειρές κειμένου. Παίρνουμε την κατακόρυφη προβολή της εικόνας, `sum(x, 2)`, και βρίσκουμε την μέγιστη τιμή την οποία θεωρούμε ως τιμή κενού μεταξύ δύο σειρών. Στην συνέχεια, διασχίζουμε την εικόνα κατακόρυφα. Όσο η προβολή της αντίστοιχης γραμμής είναι ίση με την τιμή κενού σημαίνει πως δεν υπάρχουν γράμματα. Μόλις εντοπίσουμε μια γραμμή με τιμή μικρότερη από την τιμή κενού σημαίνει πως είναι η αρχή μιας νέας σειράς κειμένου. Προσθέτουμε αυτήν και τις επόμενες γραμμές σε έναν πίνακα μέχρι να ξαναεντοπίσουμε γραμμή με προβολή ίση με την τιμή κενού. Επαναλαμβάνοντας την διαδικασία αυτή δημιουργούμε ένα `cell array` με κάθε στοιχείο i να είναι μια εικόνα με την σειρά i του κειμένου.

Χρησιμοποιώντας το `cell array` αυτό, επαναλαμβάνουμε την ίδια διαδικασία σε κάθε στοιχείο του, αλλά χρησιμοποιώντας την οριζόντια προβολή και διασχίζοντας την εικόνα οριζόντια. Με τον τρόπο αυτό βρίσκουμε τα γράμματα κάθε γραμμής και δημιουργούμε ένα `cell` με στοιχεία άλλα `cell`, ίσα στο πλήθος τους με τις γραμμές του κειμένου. Καθένα από αυτά περιέχουν εικόνες με τα γράμματα της αντίστοιχης σειράς.

Η διαδικασία αυτή γίνεται στην συνάρτηση `getChars`.

### 3.3 Διόρθωση γραμμάτων

Στα τελικά κείμενα που δόθηκαν το στάδιο αυτό δεν είναι πλήρως απαραίτητο (έως καθόλου), αλλά δημιουργήθηκε από την ανάγκη διόρθωσης της αρχικής εικόνας κειμένου, που είχε ενωμένα γράμματα και υπογραμμίσεις. Όπως αναφέρεται και στην αρχή, διατηρείται για την γενικότητα του συστήματος.

Αρχικά, κάνουμε *resize* κάθε εικόνα γράμματος σε σταθερό μέγενθος ώστε η μέγιστη διάσταση να είναι ίση με 250. Παρόμοια με τον εντοπισμό γωνίας, τα μεγέθη αυτά έχουν επιλεχθεί με δοκιμή και σφάλμα σε συνδυασμό με διάφορα μεγέθη από *structure elements*. Το *resize* είναι σημαντικό κομμάτι, αφού διαπιστώθηκε πως και να δουλέψουν καλά τα επόμενα βήματα και καταφέρουμε να βρούμε επιτυχημένα το περίγραμμα ενός γράμματος, τα *feature vectors* δύο ίδιων γραμμάτων διαφορετικού μεγέθους δεν μοιάζουν αρκετά ώστε να εντοπιστεί σωστά το γράμμα. Δηλαδή, τα *feature vectors* που εξάγουμε δεν είναι *scale invariant*.

Στην συνέχεια, θέλουμε να απαλείψουμε ενωμένα γράμματα και υπογραμμίσεις χρησιμοποιώντας μορφολογικούς τελεστές και μετά να εντοπίσουμε τα νέα διορθωμένα γράμματα. Η λογική είναι πως κάνουμε ένα οριακό *closing* (με λίγο μεγαλύτερο *structure element* θα καταστρέψουνταν πολλά γράμματα) και ξανακάλούμε την συνάρτηση εντοπισμού γραμμάτων με όρισμα μόνο την εικόνα του γράμματος στην δεδομένη επανάληψη. Η λογική που ακολουθεί δεν φαίνεται να είναι η βέλτιστη, αλλά η αρχική εικόνα εισόδου έχει πολλές οριακές περιπτώσεις και δεν υπάρχει ενιαίος τρόπος αντιμετώπισης οπότε δουλεύει πολύ καλά. Γενικά, κρατάμε το αρχικό *cell array* με τους χαρακτήρες ακέραιο και δουλεύουμε σε προσωρινές μεταβλητές μέχρι να σιγουρευτούμε πως οι αλλαγές που κάνουμε θέλουμε να εφαρμοστούν. Αυτό γίνεται κυρίως επειδή το οριακό *closing* καταστρέφει την ομαλότητα των γραμμάτων και δεν θέλουμε να κρατήσουμε τις αλλαγές παρά μόνο αν ήταν απαραίτητο για τον εντοπισμό νέων γραμμάτων. Έτσι, καλούμε την *getChars* μία με την εικόνα το γράμματος όπως εντοπίστηκε αρχικά και μία με την εικόνα μετά το *closing*.

- Έαν η *getChars* επιστρέψει λιγότερες γραμμές κειμένου απ' όσες υπήρχαν πριν τον τελεστή *close* τότε σημαίνει πως υπήρχε μια υπογράμμιση (ή θόρυβος στην γενική περίπτωση) που διαγράφηκε με τον τελεστή και έχουμε τρεις περιπτώσεις:
  1. Άμα οι καινούριες σειρές που παίρνουμε είναι 0 σημαίνει πως ήταν σκετή υπογράμμιση που προεξείχε από γράμματα και διαγράφηκε. Αφαιρούμε την αντίστοιχη εικόνα από το *cell array*.
  2. Άμα η καινούρια σειρά είναι 1, τότε διαγράφηκε η υπογράμμιση. Αντικαθιστούμε την αντίστοιχη εικόνα με την/τις καινούρια/καινούριες (πληθυντικός καθώς υπάρχει περίπτωση να υπήρχαν περισσότερα από ένα γράμματα μαζί λόγω της υπογράμμισης που τώρα χωρίστηκαν).
  3. Άμα οι καινούριες σειρές είναι 2, τότε ήταν  $i \neq j$  με υπογράμμιση οπότε κρατάμε μόνο το γράμμα χωρίς την τελεία του. Αυτό γίνεται κρατώντας την σειρά κειμένου με τα περισσότερα μαύρα *pixel*.
- Έαν η *getChars* επιστρέψει 1 γραμμή κειμένου και τις δύο φορές, έχουμε δύο περιπτώσεις.
  1. Άμα η μια αυτή γραμμή έχει έναν χαρακτήρα δεν αλλάζουμε τίποτα.
  2. Άμα έχει παραπάνω από έναν προσθέτουμε τους νέους χαρακτήρες στο *cell array*.

- Έαν επιστρέψει 2 γραμμές κειμένου έχουμε τις εξής περιπτώσεις:
  1. Έαν έχουν και οι δύο από έναν χαρακτήρα τότε είναι  $i \neq j$  και χαρακτήρας με υπογράμμιση. Κρατάμε την σειρά με τα περισσότερα μαύρα pixel.
  2. Αλλιώς είναι υπογραμμισμένοι χαρακτήρες, οπότε κρατάμε την πρώτη γραμμή και αφαιρούμε την άλλη.
- Εάν επιστρέψει 3 γραμμές είναι  $i \neq j$  υπογραμμισμένα, πιθανώς ενωμένα και με άλλους χαρακτήρες. Κρατάμε την μεσαία γραμμή και αφαιρούμε τις υπόλοιπες.

Η διαδικασία αυτή γίνεται για όλους τους αρχικούς χαρακτήρες που εντοπίστηκαν. Θα μπορούσε να ενσωματωθεί ταυτόχρονα με την αρχικό εντοπίσμο χαρακτήρων, αλλά θα χειροτέρευε αρκέτα την αναγνωσιμότητα του κώδικα.

## 4 Περιγράμματα

Για τον εντοπισμό των περιγραμμάτων αρχικά προεπεξεργαζόματε την εικόνα του γράμματος που δέχεται σαν είσοδο. Προσθέτουμε επιπλέον άσπρο περιθώριο γύρω από την εικόνα ώστε να μην υπάρξει πρόβλημα με την διόγκωση των γραμμάτων από τους μορφολογικούς τελεστές και από τους ελέγχους κατά τον εντοπισμό του περιγράμματος. Εφαρμόζουμε *opening* στην εικόνα για να εξομαλύνουμε το περίγραμμα του γράμματος και μετά *dilation* ώστε με την αφαίρεση της *opened* εικόνας να μείνει μόνο το περίγραμμα. Τέλος, κάνουμε *thinning* ώστε να έχει το περίγραμμα σίγουρα μόνο ένα pixel πάχος. Πλέον, το γράμμα μας είναι άσπρο σε μαύρο φόντο. Ο εντοπισμός του περιγράμματος είναι μια απλή επαναληπτική διαδικασία ελέγχου γειτόνων.

1. Βρίσκουμε το πιο πάνω και αριστέρα άσπρο pixel. Προσθέτουμε τις συντεταγμένες του στα στοιχεία του περιγράμματος και ξεκινάμε τον έλεγχο γειτόνων από αυτό.
2. Κάθε φορά που βρίσκουμε ένα σημείο του περιγράμματος, κάνουμε μαύρο το αντίστοιχο pixel. Ουσιαστικά αφαιρούμε το περίγραμμα από την εικόνα καθώς το διασχίζουμε. Έτσι δεν μπορεί να μπερδέψει ο αλγόριθμος καινούριο στοιχείο του περιγράμματος με παλιό. Ακόμη, μπορούμε με την ίδια διαδικασία να βρούμε το δεύτερο και το τρίτο περίγραμμα του γράμματος αφού δεν θα υπάρχουν τα προηγούμενα στην εικόνα.
3. Κάθε φορά κρατάμε μια μεταβλητή που δείχνει την κατεύθυνση στην οποία κινούμαστε (δεξιά ή αριστερά). Με αυτό τον τρόπο πολλά σημεία του περιγράμματος βρίσκονται χωρίς να χρειάζεται να ελέγχουμε για καινούριους γείτονες.
4. Επειδή ξεκινάμε αυθαίρετα να διασχίζουμε την εικόνα οριζοντίως μέχρι να βρούμε σημείο εκτός περιγράμματος, σε κάθε επανάληψη κρατάμε τις συντεταγμένες του προηγούμενου σημείου ωστέ αν βγούμε εκτός περιγράμματος να επιστρέψουμε σε αυτό και να ελέγχουμε για καινούριους γείτονες.
5. Ελέγχουμε για πιθανούς γείτονες με την παρακάτω σειρά. Εφόσον βρούμε λευκό pixel αλλάζουμε αντίστοιχα τους δείκτες ( $i$ ,  $j$ ) και την κατεύθυνση κίνησης.
  - north-east (NE): θέτουμε την κατεύθυνση, *stepN* ως 1 (δεξιά).
  - south-east (SE): θέτουμε την κατεύθυνση, *stepN* ως 1 (δεξιά).
  - south (S): θέτουμε την κατεύθυνση, *stepN* ως 1 (δεξιά).

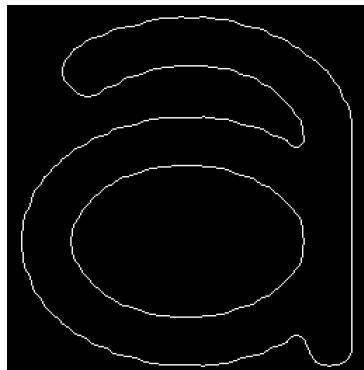
- south-west (*SW*): θέτουμε την κατεύθυνση, *stepN* ως -1 (αριστερά).
  - north-west (*NW*): θέτουμε την κατεύθυνση, *stepN* ως -1 (αριστερά).
  - north (*N*): θέτουμε την κατεύθυνση, *stepN* ως 1 (δεξιά).
6. Αν δεν βρεθεί νέος γείτονας ο αλγόριθμος σταματάει και έχουμε βρει το περίγραμμα του γράμματος.

Η διαδικασία αυτή επαναλαμβάνεται κάθε φορά που βρίσκουμε ένα νέο άσπρο *pixel* ώστε να βρούμε όλα τα περιγράμματα. Όλα τα περιγράμματα ξεκινάνε από το πάνω αριστερά *pixel*. Έχουμε θέσει ένα αυθαίρετο *threshold* των 10 *pixel*. Εάν στην εικόνα έχουν μείνει λιγότερα από 10 άσπρα σημεία τότε θεωρούμε ότι έχει γίνει λάθος με την εφαρμογή των τελεστών και τα σημεία αυτά δεν αποτελούν πραγματικό περίγραμμα. Στα Σχήματα 4.1 - 4.2 φαίνονται όλα τα στάδια μιας εικόνας που περιέχει το γράμμα *a* και τα περιγράμματα για τα *f*, *I* και *e*. Προφανώς κάποια στάδια έχουν κυριολεκτικά διαφορά κάποιων *pixel* οπότε δεν είναι πάντα ορατή η διαφορά, αλλά παίζει σημαντικό ρόλο.

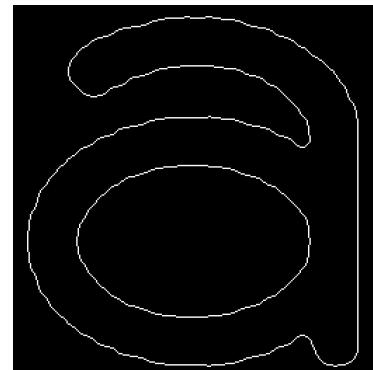
(a) Γράμμα *a*

(b) *a* opened

(c) *a* dilated

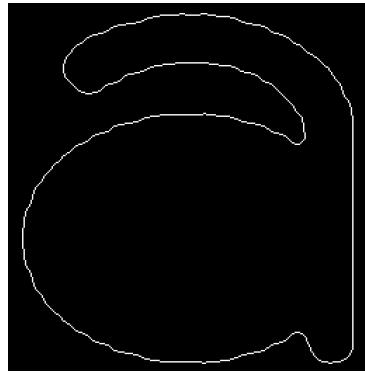


(d) *a* border

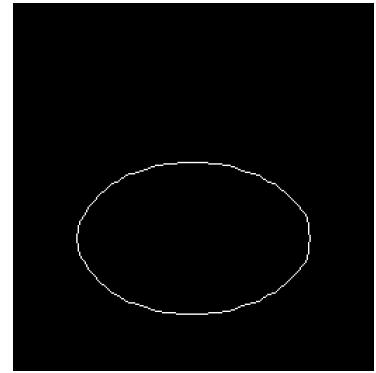


(e) *a* thinned

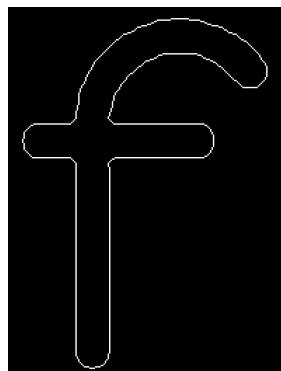
Σχήμα 4.1: Στάδια του *a* πριν τον εντοπισμό περιγράμματος



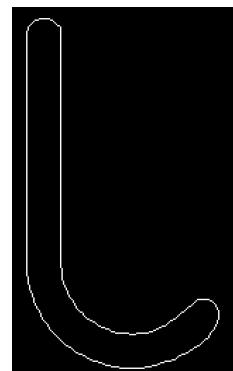
(a) Εξωτερικό περίγραμμα του a



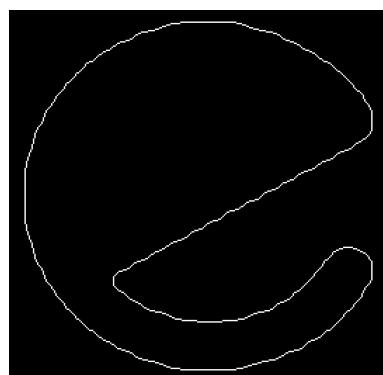
(b) Εσωτερικό περίγραμμα του a



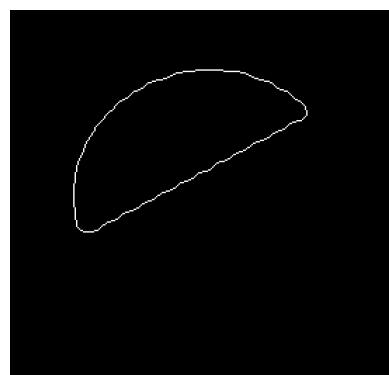
(c) Περίγραμμα του f



(d) Περίγραμμα του l



(e) Εξωτερικό περίγραμμα του e



(f) Εσωτερικό περίγραμμα του e

Σχήμα 4.2: Περιγράμματα των a, f, l και e

## 5 Εκπαίδευση

Για τον εντοπισμό των γραμμάτων εξάγουμε έναν περιγραφέα βασισμένο στον *Discrete Fourier Transform* της μιγαδικής ακολουθίας που σχηματίζεται από τις συντεταγμένες κάθε σημείου του περιγράμματος. Χρησιμοποιούμε παρεμβολή ώστε όλοι οι περιγραφές να έχουν το ίδιο μέγεθος,  $N$ , αρχικά αυθαίρετο. Αν ένα γράμμα έχει παραπάνω από ένα περίγραμμα, τότε δημιουργούμε πάλι έναν ενιαίο περιγραφέα, όπου τοποθετούμε τον *DFT* κάθε περιγράμματος στο τέλος του προηγούμενου. Σε αυτή την περίπτωση, η παρεμβολή κάθε περιγράμματος γίνεται σε ποσοστό του συνολικού μεγέθους  $N$ . Για παράδειγμα, στο γράμμα e, το 70% του περιγραφέα θα είναι από το εξωτερικό περίγραμμα και το υπόλοιπο από το εσωτερικό.

Έχοντας τους περιγραφές, χρησιμοποιούμε ως *labels* τα αντίστοιχα γράμματα σε ASCII μορφή για εποπτευόμενη μάθηση τριών *kNN classifiers* (*fitcknn*) με μετρική απόστασης την προεπιλεγμένη ευχλάδεια απόσταση που χρησιμοποιεί η συνάρτηση.

## 6 Κώδικας

Καταρχάς, να σημειωθεί πως το *demo* που παραδίδεται παίρνει αρκετή ώρα να εκτελεστεί (~2 λεπτά), κυρίως λόγω των πολλών περιστροφών που γίνονται στην συνάρτηση *findRotationAngle*. Όπως αναφέρθηκε και παραπάνω, οι χρόνοι αυτοί θα μπορούσαν να μειωθούν είτε με *vectorization* είτε μειώνοντας το εύρος στο οποίο αναζητεί γωνίες μετά την πρώτη εκτίμηση, το οποίο έχει επιλεχθεί έτσι ώστε να καλύπτει και ακραίες περιπτώσεις όπου η πρώτη εκτίμηση δεν είναι τόσο καλή.

Η διαδικασία που κάνει το *demo* έχει ως εξής:

1. Ανοιγεί την εικόνα *text1\_v3* που θα χρησιμοποιηθεί για εκπαίδευση και το αντίστοιχο κείμενο σε ASCII.
2. Καλεί την *imageLetters* η οποία εκτελεί όλη την διαδικασία που περιγράφηκε στα κεφάλαια 2-4 και επιστρέφει ένα *cell array* με τα περιγράμματα κάθε γράμματος και ένα *cell* με *cell arrays* με τα γράμματα κάθε σειράς.
3. Υπολογίζει ένα *featureSize* ανάλογα με το μέγεθος των περιγραμμάτων που βρέθηκαν και καλεί την *getDescriptor* που επιστρέφει τους περιγραφές των περιγραμμάτων που βρέθηκαν.
4. Χωρίζει περιγραφές και κείμενο σε *train* και *test* dataset με αναλογία 70:30.
5. Εκπαιδεύει τρία μοντέλα καλώντας την *train* η οποία επιστρέφει τα μοντέλα αυτά.
6. Τεστάρει τα μοντέλα με το υπόλοιπο 30% του αρχικού κειμένου. Η *test* επιστρέφει ένα *char array* με το κείμενο που πρόβλεψε το μοντέλο.
7. Επαναλαμβάνει τα βήματα 1-3 για την εικόνα *text\_2*, χρησιμοποιώντας ίδιο *featureSize* με την εικόνα 1.
8. Καλεί την *test* για την εικόνα 2.
9. Εμφανίζει τους *confusion matrices* και επιστρέφει τα *weighted accuracies* για το *test* σύνολο του πρώτου κειμένου και για όλο δεύτερο κείμενο.

10. Καλεί την `readtext` με την εικόνα 2. Η συνάρτηση δέχεται μία εικόνα κειμένου, 3 *classifying* μοντέλα και το `featureSize` που χρησιμποιήθηκε για την εκπαίδευση των μοντέλων ώστε να γίνεται παρεμβολή των περιγραφέων στο ίδιο μέγεθος. Επιστρέφει ένα `cell array` με το κείμενο κάθε γραμμής της εικόνας σε ASCII.

11. Αποθηκεύει το κείμενο που επιστρέφει η `readtext` σε ένα αρχείο `predText.txt`.

Προφανώς στην `demo` κάποια πράγματα επαναλαμβάνονται για να βγουν τα αποτελέσματα που παρουσιάζονται στο επόμενο κεφάλαιο. Στην πραγματικότητα, τα βήματα 1-5 θα γινόντουν σε μία συνάρτηση που θα αποθήκευε τα μοντέλα που εκπαίδευτηκαν και το `featureSize` σε ένα `.mat` αρχείο και θα τα φορτώναμε όποτε θέλαμε να χρησιμοποιήσουμε την `readText`.

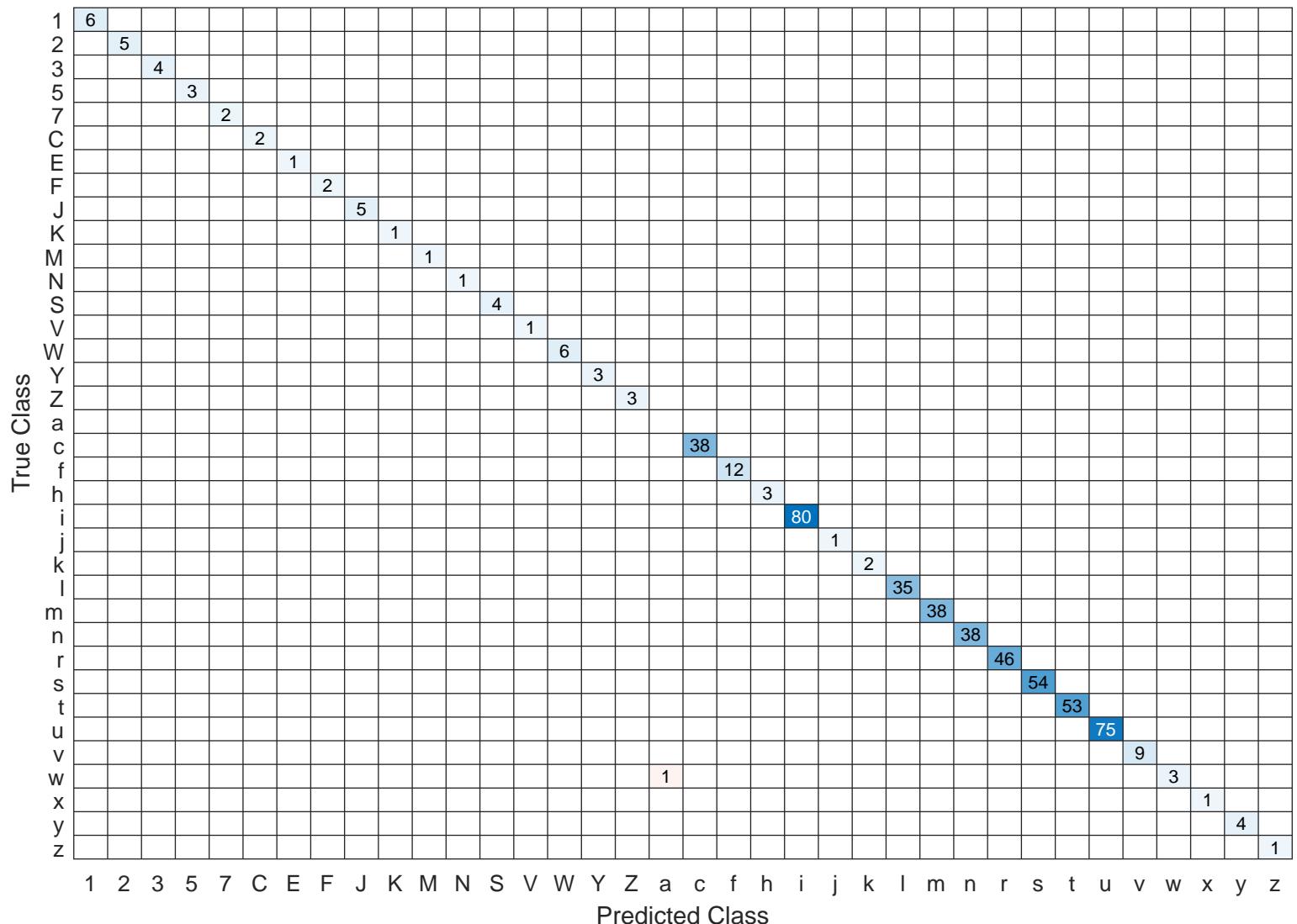
## 7 Αποτελέσματα

Για την τελική χρήση των μοντέλων αποφασίστηκε `featureSize = 100`. Η διαδικασία έγινε κυρίως με `trial and error`. Παρατηρήθηκε πως οι ακραία μεγάλες τιμές αυξάνουν την πιθανότητα για *overfitting*. Δηλαδή, εάν θέσουμε το μέγεθος όλων των περιγραφέων μετά την παρεμβολή ίσο με το μεγαλύτερο μήκος περιγράμματος ( $\sim 900$ ) τότε και στο `test dataset` του 1ου και σε όλο το 2ο έχουμε αρκετά κακά αποτελέσματα (συνολική ακρίβεια 80%). Αντίθετα, χρησιμοποιώντας το ελάχιστο μήκος (10-20) παίρνουμε τα καλύτερα αποτελέσματα και για το `test dataset` ( $\sim 99\%$ ) και για το 2ο κείμενο (97 – 98%). Αν χρησιμοποιήσουμε την διάμεσο ή τον μέσο όρο (400-500) παίρνουμε καλά αποτελέσματα και για τα δύο αλλά όχι βέλτιστα (96 – 97%), οπότε θα ήταν μια καλή "τυφλή επιλογή". Οποιαδήποτε επιλογή 10-400 μας δίνει ικανοποιητικά αποτελέσματα.

Ακόμη, την συνολική ακρίβεια επηρεάζει και το σύνολο εκπαίδευσης που διαλέγουμε. Η επιλογή έγινε δοκιμάζοντας τυχαίες μεταθέσεις του 1ου κείμενου και επιλέγοντας την καλύτερη. Η παράμετρος αυτή δεν επηρεάζει τα αποτελέσματα με τον ίδιο τρόπο στο 1ο και στο 2ο κείμενο, αλλά αντίστροφα. Συγκεκριμένα, με κατάλληλη επιλογή, η ακρίβεια στο 2ο κείμενο μπορεί να ανέβει πάνω από 98% αλλά πέφτει στο ίδιο επίπεδο και η ακρίβεια του 1ου. Ή, αντίθετα, με άλλη μετάθεση, μπορεί η ακρίβεια του 1ου κείμενου να πάει στο 100% αλλά η ακρίβεια του 2ου θα είναι 96-97%. Θεωρώντας πως η ακρίβεια που βγάζουμε για το δεύτερο κείμενο είναι πιο αντικειμενική για την γενική εικόνα του συστήματος επιλέχθηκε η μετάθεση που βελτιστοποιεί την ακρίβεια στο 2ο κείμενο.

Στα Σχήματα 7.1-7.6 φαίνονται οι πίνακες σύγχυσης για τα αρχεία `text1_v3.png` και `text2_rot.png` χωρίς σημεία στίξης. Η σταθμισμένη ακρίβεια στην πρώτη περίπτωση είναι 99.5% ενώ στην δεύτερη 97.3%. Εάν δεν αφαιρέσουμε τα σημεία στίξης έχουμε ακρίβεια 99.62% και 96.49% αντίστοιχα.

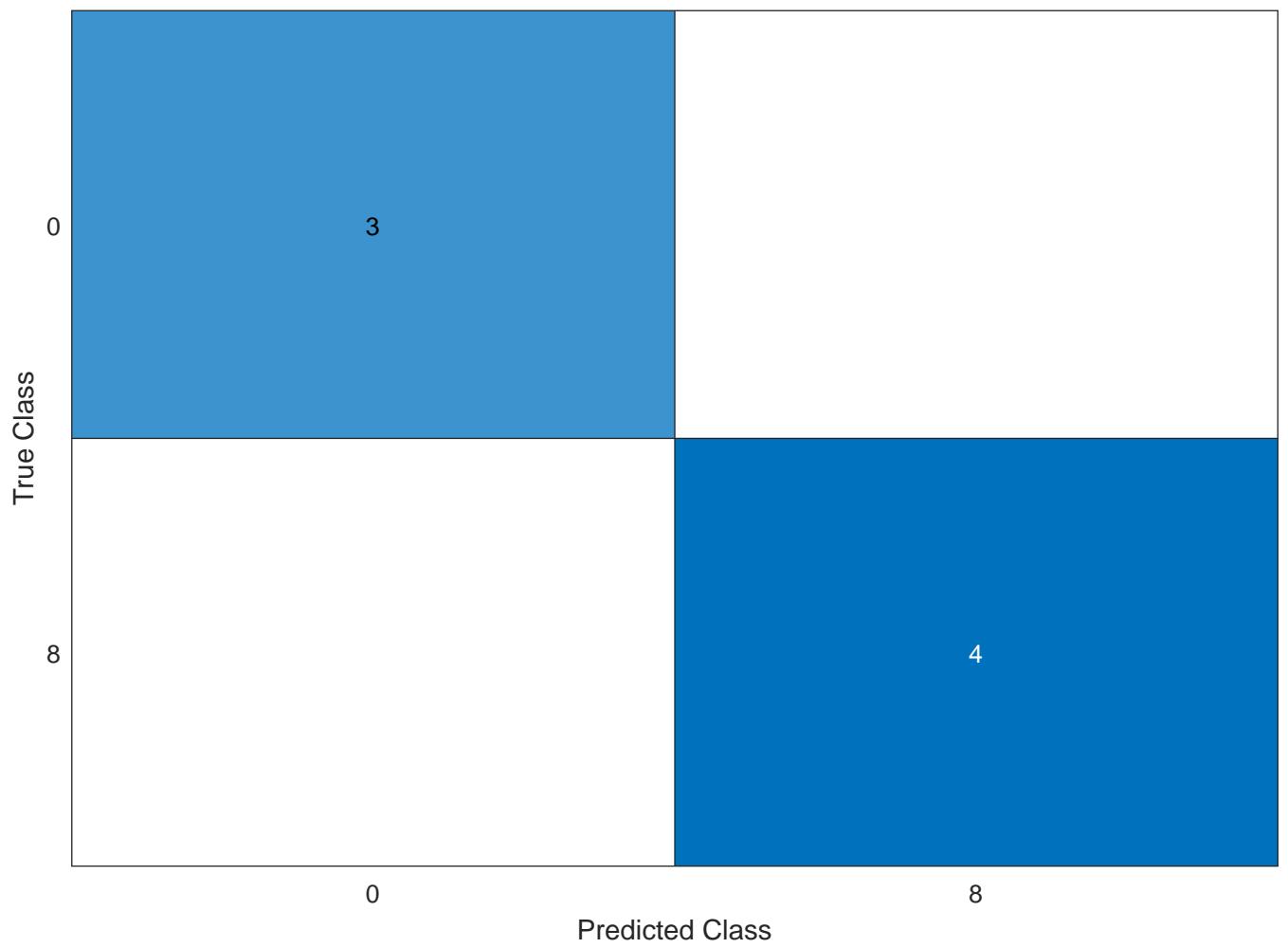
Για να αναπαραχθούν τα αποτελέσματα αυτά αρκεί κάποιος να τρέξει το `demo.m` αρχείο με τις εικόνες και τα αντίστοιχα κείμενα στον ίδιο φάκελο. Ο κώδικας αυτή την στιγμή χρησιμοποιεί το `indexes.mat` για την μετάθεση του πρώτου κειμένου που δίνει το καλύτερο σύνολο εκπαίδευσης. Αυτό μπορεί να αλλάζει και σε απλώς τυχαία μετάθεση μειώνοντας ελάχιστα την σταθμισμένη ακρίβεια.



Σχήμα 7.1: Πίνακας σύγχυσης πρώτου κειμένου για χαρακτήρες ενός περιγράμματος

		Predicted Class														
		4	6	9	A	D	P	Q	a	b	d	e	g	o	p	q
True Class	4															
	6	4														
	9		1													
	A			3												
	D				3											
	P					3										
	Q						2									
	a							48								
	b								7							
	d									23						
	e										84					
	g											7				
	o												27			
	p													17	1	
	q													2	6	

Σχήμα 7.2: Πίνακας σύγχυσης πρώτου κειμένου για χαρακτήρες δύο περιγραφμάτων



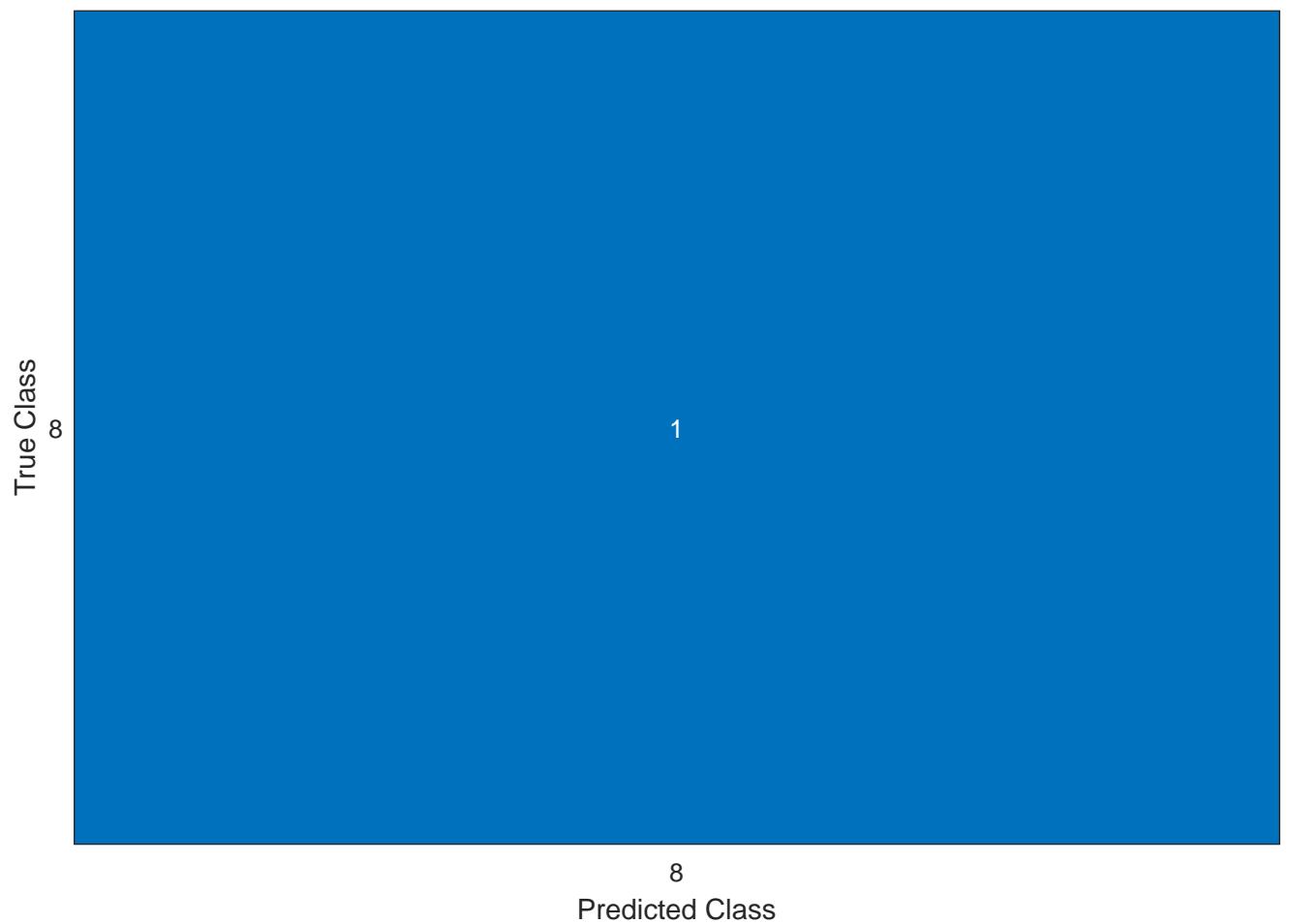
Σχήμα 7.3: Πίνακας σύγχυσης πρώτου κειμένου για χαρακτήρες τριών περιγραμμάτων

True Class	1	2	3	7	C	M	S	Y	c	f	h	i	j	k	l	m	n	r	s	t	u	v	w	x	y		
1	1																										
2		1																									
3			1																								
7				1																							
C																											
M																											
S																											
Y																											
c									2								65										
f										3								51									
h																		83									
i												6						126									
j																		16									
k																			11								
l																			2		71						
m																				32							
n																					121						
r																					139						
s		6																			114						
t			3									6									137						
u					1																41						
v								1													26						
w						1																28					
x																							3				
y																								24			

Σχήμα 7.4: Πίνακας σύγχυσης δεύτερου κειμένου για χαρακτήρες ενός περιγράμματος

	4	3										
	6											
	9		1									
a		8			183							
b						12						
d		1					79					
e								236				
g									36			
o										108		
p											33	2
q												
	4	6	9	a	b	d	e	g	o	p	q	
	Predicted Class											

Σχήμα 7.5: Πίνακας σύγχυσης δεύτερου κειμένου για χαρακτήρες δύο περιγραμμάτων



Σχήμα 7.6: Πίνακας σύγχυσης δεύτερου κειμένου για χαρακτήρες τριών περιγραμμάτων