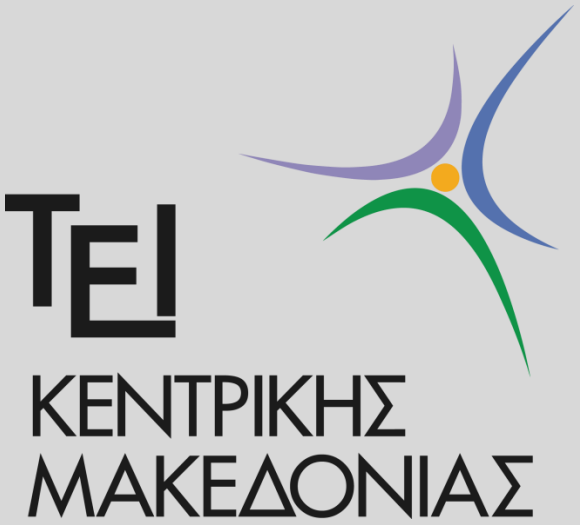


Πτυχιακή Εργασία - Νικηφόρος Αρχάκης
Α.Ε.Μ. 3621

Επιβλέπων Καθηγητής: Αθανάσιος Νικολαΐδης

Εξαγωγή κειμένου από σύνθετες σκηνές εικονοσειρών

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να εκφράσω τις ευχαριστίες μου στον Αναπληρωτή Καθηγητή Δρ. Αθανάσιο Νικολαΐδη, για τη στήριξη και την υπομονή του, αλλά κυρίως για την δυνατότητα που μου έδωσε να εκπονήσω την παρούσα πτυχιακή εργασία, η οποία αποδείχθηκε απαιτητική, καινοτόμα και πολύ ενδιαφέρουσα, πράγμα που συνέβαλε άμεσα στην προσωπική μου εξέλιξη αλλά και στην πυροδότηση της επερχόμενης καριέρας μου ως Μηχανικός Πληροφορικής.

Επιπλέον, ευχαριστώ θερμά όλους τους καθηγητές του τμήματός μου, για την καλοσύνη τους και την εξαίρετη διδασκαλία τους όλα αυτά τα χρόνια. Να ευχαριστήσω επίσης την διοίκηση του ιδρύματος και την αγαπητή γραμματεία του τμήματος, για την μεγάλη βοήθεια που προσέφεραν σε εμένα αλλά και που προσφέρουν συνεχώς σε όλους τους σπουδαστές.

Τέλος, να εκφράσω την αγάπη και την ευγνωμοσύνη μου στην οικογένειά μου, στους κοντινούς ανθρώπους και φίλους, για την ασταμάτητη στήριξη που μου παρείχαν.

ΠΕΡΙΕΧΟΜΕΝΑ

1 ΕΙΣΑΓΩΓΗ

1.1 ΨΗΦΙΑΚΗ ΕΠΕΞΕΡΓΑΣΙΑ ΕΙΚΟΝΑΣ

Η Ψηφιακή Επεξεργασία Εικόνας (ΨΕΕ) είναι αναμφισβήτητα ένας από τους πιο επαναστατικούς και συναρπαστικούς τομείς της Πληροφορικής στις μέρες μας. Καθώς η ΨΕΕ είναι άρτια συνδεδεμένη με την Τεχνητή Νοημοσύνη και την Μηχανική Όραση, αντιλαμβάνεται κανείς τον κρίσιμο και σπουδαίο ρόλο που κατέχει στην ανάπτυξη της ήδη υπάρχουσας αλλά και της επερχόμενης επαναστατικής τεχνολογίας, η οποία πρόκειται να μεταμορφώσει ριζικά τον σύγχρονο πολιτισμό μας.

Ένα κύριο χαρακτηριστικό της ΨΕΕ, είναι ότι η εφαρμογή της παρατηρείται παράλληλα σε πολλές και διαφορετικές επιστήμες. Μερικές από αυτές είναι η Αστρονομία, η Διαστημική, η Ιατρική, η Βιολογία και η Ρομποτική. Ουσιαστικά, η ΨΕΕ εφαρμόζεται σε οτιδήποτε καλείται να αποκτήσει «όραση» (*Μηχανική Όραση*), με σκοπό την αναγνώριση και κατανόηση κειμένων, προσώπων ή αντικειμένων, όπως για παράδειγμα τα επερχόμενα αυτόνομα οχήματα χωρίς οδηγό (*driverless cars*).

1.2 ΠΡΟΒΛΗΜΑΤΑ ΠΟΥ ΑΠΑΙΤΟΥΝ ΛΥΣΗ

Όπως είδαμε, η ΨΕΕ είναι ένας εξαιρετικά δυναμικός και σημαντικός επιστημονικός κλάδος με πληθώρα εφαρμογών σε διάφορους τομείς. Αυτό εξηγεί και το γεγονός ότι διεκπεραιώνονται συνεχώς, πολλές και σημαντικές έρευνες σε αυτόν τον τομέα από διάφορα πανεπιστήμια, οργανισμούς και εταιρίες ανά τον κόσμο. Οι έρευνες αυτές μελετούν κάποιες από τις σημαντικότερες και δυσκολότερες προκλήσεις του τομέα αυτού. Πολλά από τα προβλήματα αυτά, αφορούν πτυχές της Μηχανικής Μάθησης, όπου στόχος είναι, ο υπολογιστής να αποκτήσει ή να βελτιώσει την ικανότητα του να αναγνωρίζει και να κατανοεί το περιεχόμενο μιας εικόνας. Πιο συγκεκριμένα, ένα αρκετά ενδιαφέρον αλλά και κρίσιμο κομμάτι των προκλήσεων που προαναφέρθηκαν, είναι η ικανότητα του υπολογιστή να αναγνωρίζει και να ερμηνεύει με επιτυχία, το κείμενο που τυχόν να εμπεριέχεται σε ψηφιακά πολυμεσικά αρχεία.

Αυτή η ικανότητα είναι εξαιρετικά σημαντική καθώς οι απαιτήσεις της σύγχρονης τεχνολογίας αυξάνονται συνεχώς. Ως παράδειγμα, ας θυμηθούμε τα *driveless cars*. Είναι απαραίτητο, ένα τέτοιου είδους αυτοκίνητο, να μπορεί να εντοπίζει άμεσα αλλά και να διαβάζει με ορθότητα τις επιγραφές που αναγράφονται στις οδικές επισημάνσεις του αυτοκινητοδρόμου.

1.3 ΣΤΟΧΟΣ ΤΗΣ ΠΤΥΧΙΑΚΗΣ

Ο στόχος την πτυχιακής μπορεί να διαιρεθεί σε δύο σκέλη. Το πρώτο αφορά την έρευνα αλγορίθμων, μεθοδολογιών και τεχνικών για τον εντοπισμό και την εξαγωγή ενσωματωμένου κειμένου σε σκηνές εικονοσειρών (βίντεο) με περίπλοκο φόντο. Το δεύτερο σκέλος, αφορά την προγραμματιστική υλοποίηση και εφαρμογή της παραπάνω σύλληψης, με στόχο την ανάπτυξη ενός ολοκληρωμένου και υψηλού επιπέδου λογισμικού με οπτικοποιημένο περιβάλλον χρήσης.

Η παρούσα πτυχιακή συνδυάζει μεθόδους και τεχνικές από διάφορα επιστημονικά άρθρα τα οποία εξετάζουν χρησιμοποιώντας διάφορες προσεγγίσεις και τεχνικές, το πρόβλημα με το οποίο είμαστε αντιμέτωποι. Επίσης, όπως θα δούμε και στο Κεφάλαιο 3, η συγγραφή υψηλής ποιότητας κώδικα αλλά και η χρήση προχωρημένων τεχνολογιών λογισμικού, επηρεάζουν άμεσα την ποιότητα και την απόδοση της εφαρμογής.

2 ΘΕΩΡΗΤΙΚΗ ΑΝΑΛΥΣΗ

2.1 ΣΥΝΟΠΤΙΚΗ ΠΑΡΟΥΣΙΑΣΗ ΜΕΘΟΔΟΛΟΓΙΑΣ

Σε αυτό το σημείο παρουσιάζεται συνοπτικά η μεθοδολογία και τα βήματα που ακολουθήθηκαν, ώστε να γίνει κατανοητή η προσέγγιση και στη συνέχεια να πραγματοποιηθεί η επιμέρους ανάλυση της κάθε μεθόδου. Το πρώτο βήμα που θα πρέπει να κάνουμε είναι να αποσπάσουμε έναν αριθμό πλαισίων (*frames*) από το βίντεο μας με μια συγκεκριμένη συχνότητα. Η συχνότητα αυτή, είναι ένα πλαίσιο ανά πέντε, καθώς θεωρούμε αδύνατον να αλλάξει ένα ευανάγνωστο κείμενο του βίντεο, με ταχύτερη συχνότητα από αυτή. Για κάθε πλαίσιο που αποσπάμε, εφαρμόζουμε σε αυτό, τα εξής βήματα:

Εντοπισμός Κειμένου

- Χαμηλοπερατό φίλτρο εξομάλυνσης Gauss
- Μετατροπή σε κλίμακα του γκρίζου
- Μάσκα Laplace
- Μέγιστη Διαφορά Κλίσης
- Απλή Δυαδική Κατωφλίωση
- Μορφολογική Διαστολή
- Εύρεση συνεκτικών χωρίων
- Φιλτράρισμα χωρίων βάσει των διαστάσεών τους
- Φιλτράρισμα χωρίων βάσει της πρόβλεψης του SVM μοντέλου

Εξαγωγή Κειμένου

- Αποκοπή εντοπισμένων περιοχών κειμένου
- Τεχνική Unsharp Masking
- Δυαδική κατωφλίωση του Otsu
- Τεχνική OCR
- Προγραμματιστικό φιλτράρισμα αποτελεσμάτων

2.2 ΦΙΛΤΡΟ ΕΞΟΜΑΛΥΝΣΗΣ GAUSS

Η εξομάλυνση (*smoothing*) ή αλλιώς θόλωμα (*blurring*) είναι μια πολύ βασική και χρήσιμη τεχνική που χρησιμοποιείται συχνά στην ΨΕΕ. Υπάρχουν διάφοροι λόγοι για τους οποίους θα μπορούσε κανείς να χρησιμοποιήσει αυτήν την τεχνική. Στην περίπτωσή μας, επιδιώκουμε να πετύχουμε την μείωση του θορύβου που υπάρχει στην εικόνα μας. Υπάρχουν διάφορα φίλτρα τα οποία επιτυγχάνουν αυτόν τον σκοπό. Το Gaussian φίλτρο [4] είναι ένα από τα πιο δημοφιλή αλλά και ένα από τα πιο αποτελεσματικά. Για να το εφαρμόσουμε στην εικόνα μας, αρκεί για κάθε pixel P(x,y) να υπολογίσουμε την gaussian τιμή του:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{-x^2+y2}{2\sigma^2}}$$

όπου,

x, y : οι συντεταγμένες του pixel

σ : η τυπική απόκλιση της κανονικής κατανομής

Εικόνα 2.1 - Αρχικό πλαίσιο

Εικόνα 2.2 - Εξομάλυνση Gauss

2.3 ΜΕΤΑΤΡΟΠΗ ΣΕ ΚΛΙΜΑΚΑ ΤΟΥ ΓΚΡΙΖΟΥ

Το επόμενο βήμα μας είναι να μετατρέψουμε την φιλτραρισμένη εικόνα μας, η οποία περιγράφεται στον χρωματικό χώρο RGB (Red Green Blue), στην κλίμακα του γκρίζου. Αυτό θα διευκολύνει την περαιτέρω επεξεργασία μας, καθώς τα κανάλια χρώματος θα μειωθούν από τρία σε ένα. Υπάρχουν τρείς βασικοί αλγόριθμοι [27] οι οποίοι πετυχαίνουν αυτήν την μετατροπή.

1. Βάσει της απαλότητας (*lightness*):

   Για κάθε RGB pixel υπολογίζεται η Gray τιμή του ως εξής:

$$I(x,y) = \frac{\max\big(R(x,y), G(x,y), B(x,y)\big) + \min\big(R(x,y), G(x,y), B(x,y)\big)}{2}$$

   όπου,

   x, y : οι συντεταγμένες του RGB pixel

   R(x,y) : οι τιμή του καναλιού RED του RGB pixel

   G(x,y) : οι τιμή του καναλιού GREEN του RGB pixel

   B(x,y) : οι τιμή του καναλιού BLUE του RGB pixel

   Οι min και max συναρτήσεις υπολογίζουν την μικρότερη και μέγιστη τιμή των παραμέτρων τους.

2. Βάσει του μέσου όρου (*average*):

   Για κάθε RGB pixel υπολογίζεται η gray τιμή του ως εξής:

$$\frac{I(x,y) = \big(R(x,y) + G(x,y) + B(x,y)\big)}{3}$$

3. Βάσει της φωτεινότητας (*luminosity*):

   Η μέθοδος αυτή, η οποία είναι και αυτή που επιλέξαμε, εκμεταλλεύεται το γεγονός ότι το ανθρώπινο μάτι είναι πιο ευαίσθητο στο πράσινο χρώμα σε σχέση με τα υπόλοιπα. Έτσι, για κάθε RGB pixel υπολογίζεται η gray τιμή του ως εξής:

$$I(x,y) = 0.21R(x,y) + 0.72G(x,y) + 0.07B(x,y)$$

Εικόνα 2.3 – Κλίμακα χρωμάτων του γκρίζου

2.4 ΜΑΣΚΑ LAPLACE

Σε αυτό το σημείο, μπορούμε να προχωρήσουμε στον ορισμό των γενικών υποψήφιων περιοχών κειμένου. Για να το πετύχουμε αυτό, ως πρώτο βήμα θα εφαρμόσουμε την Laplacian μάσκα στην εικόνα μας. Η εφαρμογή της μάσκας αυτής εντοπίζει τις περιοχές της εικόνας που υπόκεινται σε απότομες αλλαγές της έντασης των pixels (*pixel intensity*), γεγονός που την καθιστά ιδανική για τον εντοπισμό γραμμάτων στην εικόνα μας. Πριν ορίσουμε την μάσκα Laplace θα πρέπει να ορίσουμε την Laplacian μιας εικόνας ως εξής [28]:

$$L(x,y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

όπου,

x, y : οι συντεταγμένες του pixel

I(x,y) : η ένταση χρώματος

Η ένταση χρώματος κάθε pixel μπορεί να βρεθεί υπολογίζοντας το χρωματικό ιστόγραμμα (*color histogram*) της εικόνας. Από την παραπάνω μαθηματική σχέση απορρέουν ορισμένοι πυρήνες (*kernels*), οι οποίοι αντιπροσωπεύουν την σχέση αυτή στον δυσδιάστατο χώρο και μπορούν εύκολα να εφαρμοστούν προγραμματιστικά στη διακριτή δυσδιάστατη εικόνας μας. Ένας από του πιο δημοφιλείς πυρήνες είναι ο εξής:

0	-1	0
-1	4	-1
0	-1	0

Μετατοπίζοντας την παραπάνω μάσκα κατά ύψος και πλάτος επάνω στην εικόνα μας, πολλαπλασιάζουμε τους συντελεστές του πυρήνα με τα αντίστοιχα pixels και έπειτα προσθέτουμε τα γινόμενα αυτά ώστε να λάβουμε την Laplacian τιμή για το pixel που αντιστοιχεί στο κέντρο του πυρήνα.

Εικόνα 2.4 – Εφαρμογή της μάσκας Laplace

2.5 ΜΕΓΙΣΤΗ ΔΙΑΦΟΡΑ ΚΛΙΣΗΣ

Η εφαρμογή της μάσκας Laplace που μόλις πραγματοποιήσαμε, θα μας χρησιμεύσει ιδιαίτερα, καθώς έχει παρατηρηθεί [14], ότι τα pixel των περιοχών μιας εικόνας που αναπαριστούν κείμενο, παρουσιάζουν μια μεγάλη κύμανση στις Laplacian τιμές τους. Για να εκμεταλλευτούμε αποτελεσματικά το χαρακτηριστικό αυτό, ορίζουμε τον τελεστή Μέγιστης Διαφοράς Κλίσης (*MGD*) ως εξής [14]:

Αρχικά προσπελαύνουμε την εικόνα μας βάσει ενός μονοδιάστατου παραθύρου $1 \times \mathrm{N}$, όπου N περιττός αριθμός (στην υλοποίηση της εφαρμογής, οι υπολογισμοί έγιναν με $\mathrm{N} = 15$). Αντικαθιστούμε την Laplacian τιμή κάθε pixel με την MGD τιμή του, η οποία ορίζεται ως εξής:

$$MGD(x, y) = \max\big(L(x, y + k)\big) - \min(L(x, y + k))$$

όπου,

x, y : οι συντεταγμένες του pixel της εικόνας

L(x,y) : η Laplacian εικόνα

$k \in \left[-\frac{N-1}{2}, \frac{N-1}{2}\right]$, όπου N: το μήκος του $1 \times \mathrm{N}$ διανύσματος

Ως λογικό επακόλουθο, οι περιοχές κειμένου τυπικά θα έχουν μεγαλύτερες MGD τιμές από αυτές των περιοχών μη-κειμένου. Έτσι, μπορούμε σε αυτό το σημείο να κάνουμε μια πρώτη εκτίμηση για τον εντοπισμό των περιοχών κειμένου στο βίντεο.

Εικόνα 2.5 - Μέγιστη Διαφορά Κλίσης

2.6 ΔΥΑΔΙΚΗ ΚΑΤΩΦΛΙΩΣΗ

Σκοπός μας σε αυτό το σημείο είναι να μετατρέψουμε την Εικόνα 2.5 σε μια δυαδική εικόνα με δύο μόνο τιμές: 0 (μαύρο) και 255 (λευκό). Υπάρχουν πολλοί τρόποι για να το πετύχουμε αυτό. Η πιο απλή τεχνική, η οποία είναι κι αυτή που εφαρμόσαμε σε αυτό το σημείο, είναι η **απλή κατωφλίωση** (*simple thresholding*) [3], δηλαδή να θέσουμε έναν αριθμό κατώφλι, έτσι ώστε οι τιμές των pixel που ξεπερνούν το κατώφλι αυτό θα χρωματίζονται με άσπρο ενώ οι υπόλοιπες με μαύρο. Επιλέξαμε το κατώφλι με την τιμή 80.

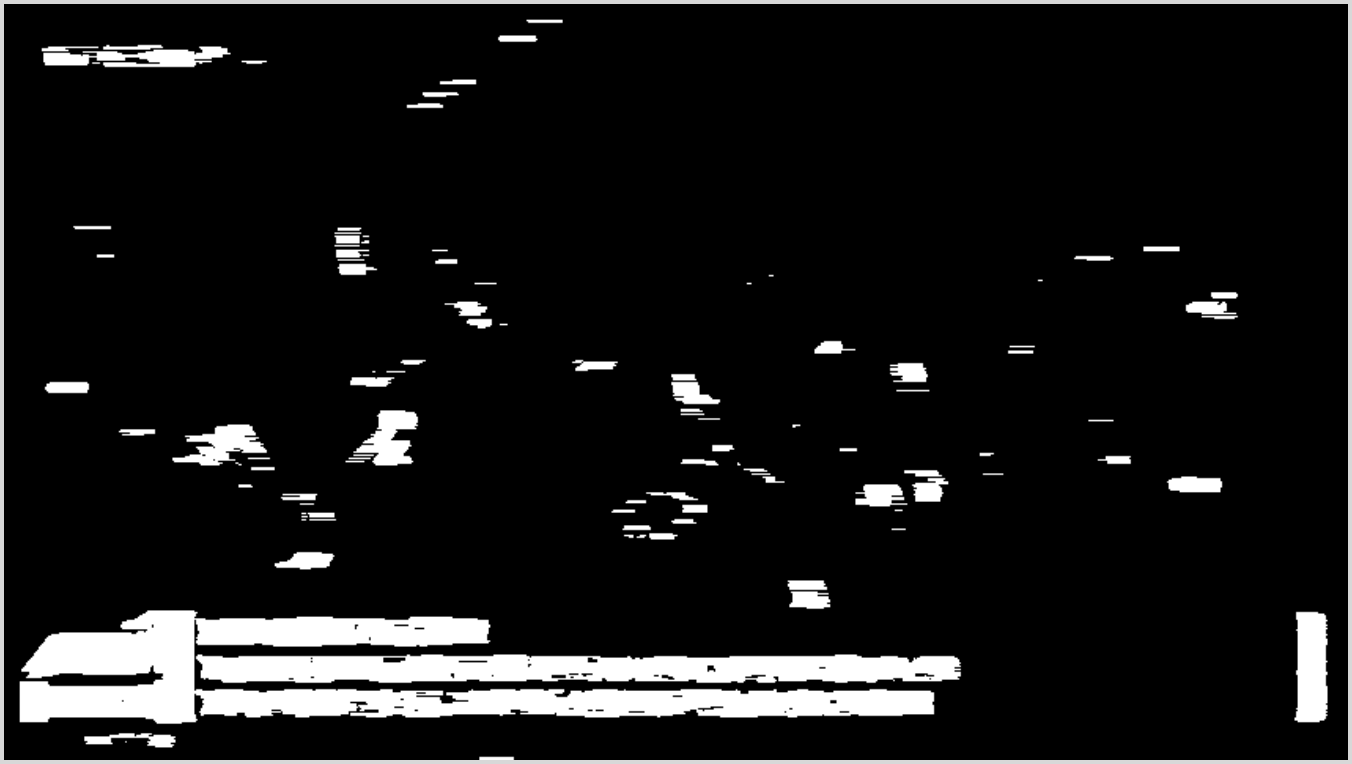
Υπάρχει και η δυνατότητα ωστόσο, να μην θέσουμε κάποιο κατώφλι αλλά αντί αυτού να επιλέξουμε κάποιον αλγόριθμο να παράξει ένα για εμάς. Αυτή η τεχνική κατωφλίωσης ονομάζεται **προσαρμοσμένη κατωφλίωση** (*adaptive thresholding*) και μερικοί αλγόριθμοι που την υλοποιούν, είναι μεταξύ άλλων, του Otsu [2] και των Μέσων Τιμών [1].

Αξίζει να αναφέρουμε και μια ακόμη τεχνική κατωφλίωσης, η οποία είναι αρκετά ισχυρή και ενδιαφέρουσα, αλλά ίσως αρκετά περίπλοκη και κοστοβόρα. Η τεχνική αυτή βασίζεται στον αλγόριθμο **συσταδοποίησης κ-μέσων** (*k-means clustering*) [5], ο οποίος χρησιμοποιείται για την συσταδοποίηση

πολυδιάστατων δεδομένων. Όταν αυτός εφαρμοστεί όμως σε μια εικόνα ώστε να παράξει μονάχα δύο συστάδες, τότε μπορεί να λειτουργήσει και αυτός, ως ένας αλγόριθμος δυαδικής κατωφλίωσης (*δυαδική κατωφλίωση κ-μέσων*) [13], όπου η μία συστάδα θα αναπαριστά τον αριθμό pixel 0 (μαύρο) και η άλλη συστάδα τον αριθμό 255 (λευκό). Εάν τα δεδομένα μας είναι και μονοδιάστατα, τότε η ειδική αυτή υποπερίπτωση των k-μέσων ονομάζεται *Jenks natural breaks optimization [8]*.

Παρακάτω φαίνονται τα αποτελέσματα της εφαρμογής της απλή δυαδικής κατωφλίωσης στην Εικόνα 2.5, με τιμή κατωφλίου 80:

Εικόνα 2.6 - Απλή δυαδική κατωφλίωση



2.7 ΜΟΡΦΟΛΟΓΙΚΗ ΔΙΑΣΤΟΛΗ

Για να παράξουμε αποτελεσματικά τις πρώτες υποψήφιες περιοχές κειμένου από την

Εικόνα *2.6*, μπορούμε να εφαρμόσουμε τον μορφολογικό τελεστή της Διαστολής (*Dilation*) στη δυαδική μας εικόνα. Την πράξη της Διαστολής την ορίζουμε ως εξής [6]:

$I \oplus S$

όπου,

I : η αρχική εικόνα μας

S : το δομικό μας στοιχείο (*structuring element*)

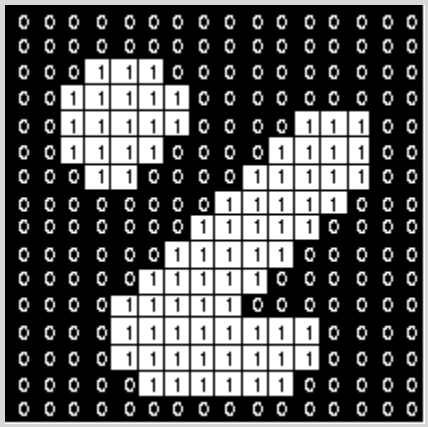
Το δομικό στοιχείο συνήθως είναι ένα τετραγωνικός πίνακας περιττών διαστάσεων που αποτελείται από 0 και 1. Επίσης, ορίζεται και η άγκυρα (*anchor*) του δομικού στοιχείου, η οποία συνήθως είναι το κεντρικό pixel, χωρίς βέβαια αυτό να είναι δεσμευτικό. Στην συνέχεια, μετατοπίζουμε με βήμα του ενός pixel το δομικό στοιχείο κατά μήκος και πλάτος της δυαδικής εικόνας και ελέγχουμε αν υπάρχει τουλάχιστον ένα pixel με τιμή 1 του δομικού στοιχείου που να επικαλύπτει κάποιο pixel με τιμή 1 ή 255 (λευκό) της δυαδικής εικόνας. Εάν ισχύει αυτό, τότε το pixel της εικόνας με το οποίο συμπίπτει η άγκυρα του δομικού στοιχείου την δεδομένη εκείνη στιγμή, χρωματίζεται με το χρώμα που έχει το προσκήνιο της εικόνας (στην περίπτωση μας το λευκό).

Παρακάτω παρουσιάζεται ένα παράδειγμα εφαρμογής της μορφολογικής διαστολής σε μια δυαδική εικόνα:

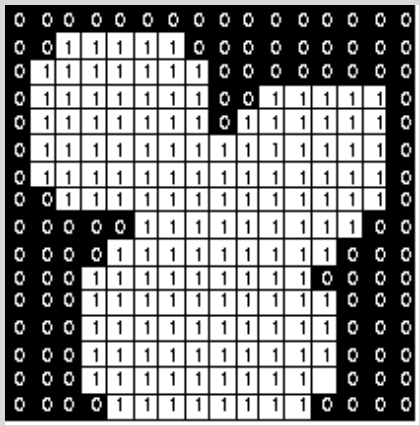
Δομικό στοιχείο

1	1	1
1	1	1
1	1	1

Εικόνα 2.7 - Αρχική δυαδική εικόνα

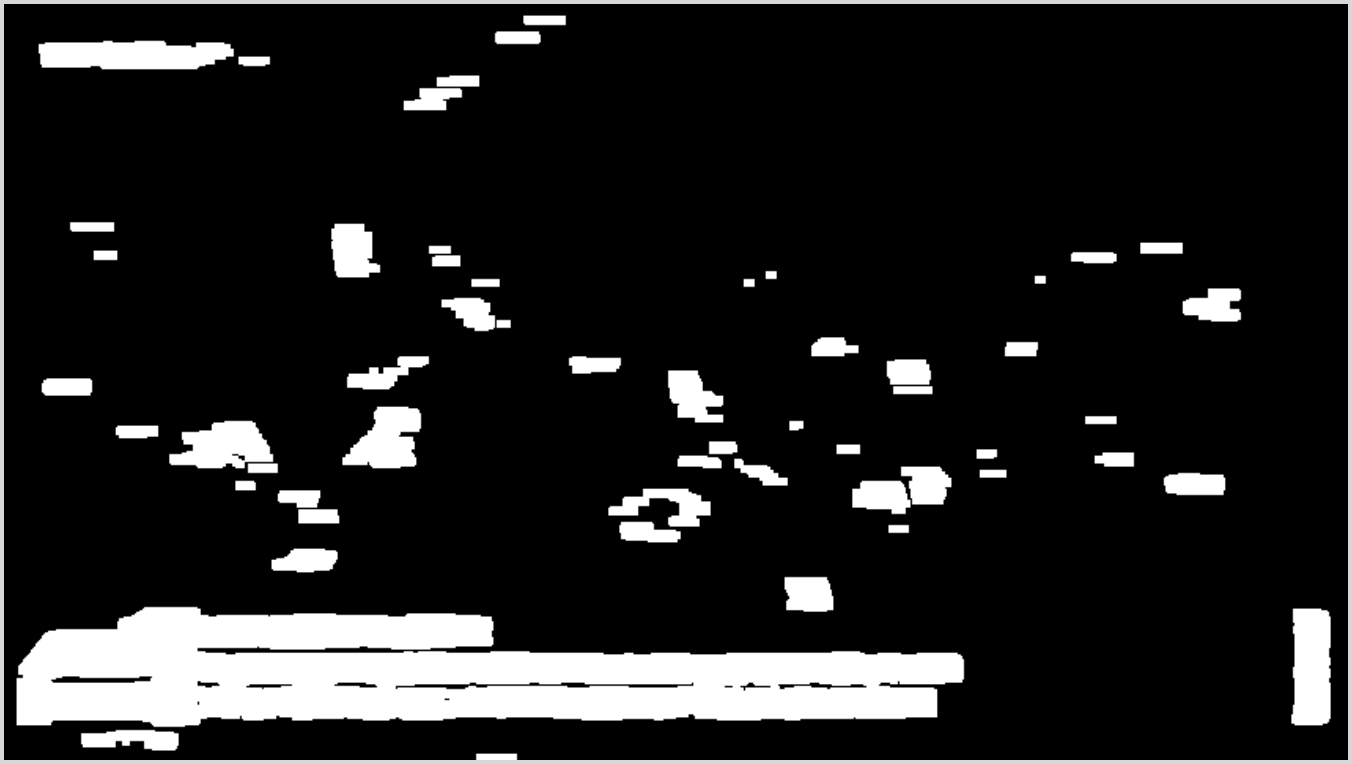


Εικόνα 2.8 - Αποτέλεσμα Διαστολής



Στην υλοποίηση της εφαρμογής χρησιμοποιήσαμε ως δομικό στοιχείο έναν πυρήνα 13×13 διαστάσεων με άγκυρα το κέντρο του και όλα τα pixel του με την τιμή 1. Παρακάτω φαίνεται το αποτέλεσμα:

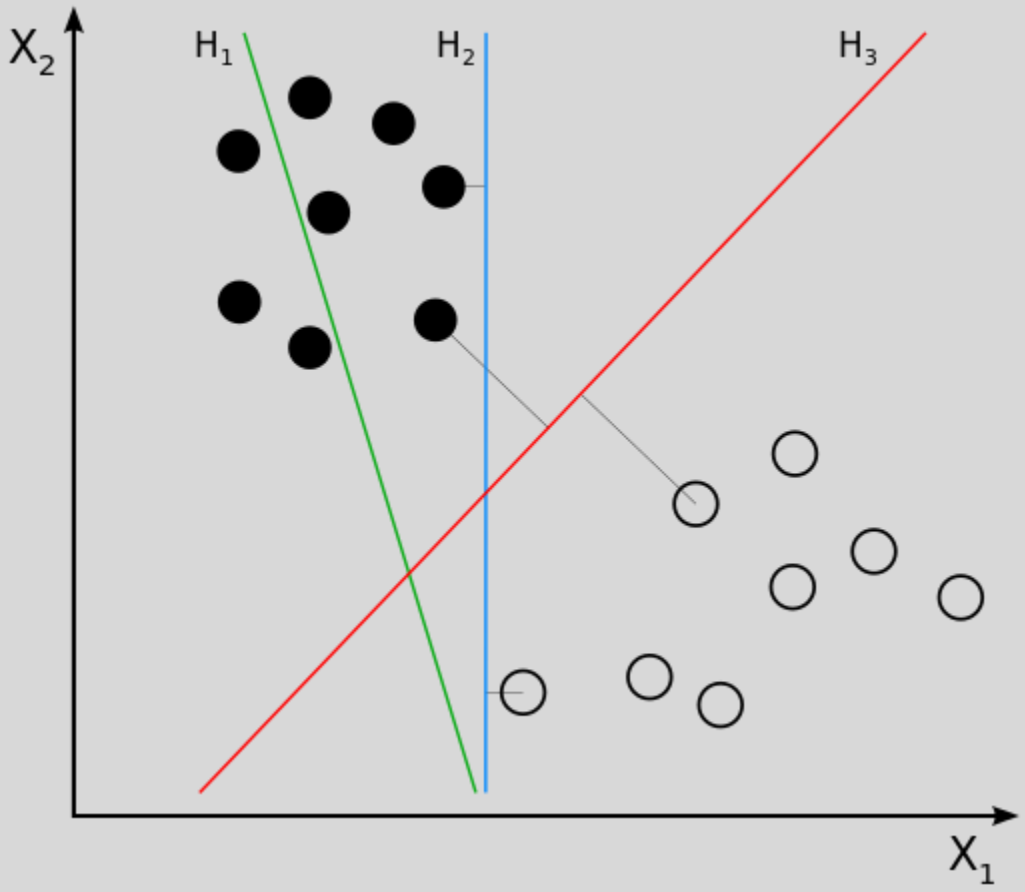
Εικόνα 2.9 - Μορφολογική Διαστολή



2.8 SUPPORT VECTOR MACHINE

Τα Support Vector Machines (*SVM*) [12] ανήκουν στην κατηγορία εκπαίδευσης με επόπτη και συνήθως χρησιμοποιούνται σε προβλήματα κατηγοριοποίησης. Παρέχοντας για κάθε κατηγορία ένα σύνολο παραδειγμάτων, με το κάθε σύνολο μαρκαρισμένο αριθμητικά ανάλογα με την κατηγορία στην οποία ανήκει, τα SVM εκπαιδεύονται και παράγουν μοντέλα τα οποία αντιπροσωπεύουν τα παραδείγματα ως σημεία στον χώρο, έτσι ώστε τα σημεία διαφορετικών κατηγοριών να απέχουν όσο το δυνατόν περισσότερο μεταξύ τους. Ένα σύνολο παραδειγμάτων (*dataset*) αποτελείται από N παραδείγματα, όπου το κάθε παράδειγμα έχει την μορφή ενός πολυδιάστατου διανύσματος (*feature vector*), με τα στοιχεία αυτού του διανύσματος, να είναι τα χαρακτηριστικά (*features*) του παραδείγματος. Αφού έχει ολοκληρωθεί η εκπαίδευση και έχει οριστεί το «όριο απόφασης», δηλαδή το υπερεπίπεδο (*γραμμικό κέλυφος*) ή η υπερεπιφάνεια (*μη-γραμμικό κέλυφος*) [30] που διαχωρίζει τα πολυδιάστατα σημεία παραδειγμάτων στον χώρο, τότε μπορούμε να τροφοδοτήσουμε καινούργια παραδείγματα στο μοντέλο, τα οποία θα μεταφραστούν και αυτά ως νέα σημεία στο χώρο. Έπειτα, μετριέται σε ποια μεριά (κατηγορία) του ορίου απόφασης βρίσκεται το καινούργιο σημείο,

επιστρέφοντας τον αριθμό-μαρκάρισμα αυτής. Η διαδικασία αυτή ονομάζεται «πρόβλεψη μοντέλου» και είναι αυτή που μας δείχνει σε ποια κατηγορία ανήκει το καινούργιο παράδειγμα που τροφοδοτήσαμε. Στο παρακάτω γράφημα βλέπουμε τρείς διαφορετικές περιπτώσεις κατηγοριοποίησης, για δύο σύνολα παραδειγμάτων των δύο διαστάσεων. Το όριο H1 έχει αποτύχει να διαχωρίσει τα δύο σύνολα παραδειγμάτων, ενώ το H2 έχει πετύχει. Παρ ’όλα αυτά, την βέλτιστη λύση, με την μεγαλύτερη απόσταση, αναπαριστά το H3.



Ρύθμιση μοντέλου (*model tuning*)

Εκπαιδεύσαμε ένα SVM μοντέλο με τις παρακάτω ρυθμίσεις και παραμέτρους:

- μίας κλάσης (*one-class SVM*)
- γραμμικό κέλυφος πυρήνα (*linear kernel*)
- παράμετρος nu, ανώτατο όριο κατηγοριοποίησης (%) : 0.01

Όπως θα αναλύσουμε και παρακάτω, σκοπός μας είναι να κατηγοριοποιήσουμε περιοχές της εικόνας με βάση την ύπαρξη ή απουσία κειμένου στις περιοχές αυτές, επομένως μιλάμε για δύο κατηγορίες. Η κλασσική προσέγγιση θα ήτανε λοιπόν, να εκπαιδεύσουμε το μοντέλο μας με δύο σύνολα παραδειγμάτων, ένα που να αποτελείται από εικόνες κειμένου και ένα χωρίς. Αυτομάτως αντιλαμβανόμαστε το πρόβλημα που προκύπτει ως προς τον ορισμό μιας εικόνας «χωρίς κείμενο». Αντιθέτως, εικόνες «με κείμενο» είναι ξεκάθαρα ορισμένες και μπορούν εύκολα να βρεθούν σε πολλά διαθέσιμα σύνολα παραδειγμάτων. Για παράδειγμα, το πρόβλημα θα ήταν πολύ απλούστερο στην περίπτωση κατηγοριοποίησης εικόνων σκύλων και γατών, καθώς μπορούμε να βρούμε εύκολα, ποικίλα και ακριβή σύνολα εικόνων για κάθε μία από τις κατηγορίες αυτές. Για τον λόγο αυτό, εφαρμόσαμε την κατηγοριοποίηση τύπου **μίας κλάσης**, η οποία μας αποδεσμεύει από το πρόβλημα αυτό, καθώς η εκπαίδευση του μοντέλου θα γίνει μόνο με ένα σύνολο παραδειγμάτων, το οποίο θα αποτελείται μόνο από εικόνες «με κείμενο». Ο SVM μίας κλάσης ανήκει στην κατηγορία «ανίχνευσης ανωμαλίας» (*novelty detection, anomaly detection ή outlier detection*) [10], όπου το μοναδικό σύνολο παραδειγμάτων μας, θεωρείται ότι ανήκει στην θετική κατηγορία και κάθε καινούργιο παράδειγμα ελέγχεται κατά πόσο ανήκει στην κατηγορία αυτή ή όχι.

Επιλέξαμε τον **γραμμικό πυρήνα** λόγω της ταχύτητας που μας προσφέρει στην εκπαίδευση και στην πρόβλεψη του μοντέλου και της ελάχιστης ρύθμισης των παραμέτρων που απαιτεί, σε αντίθεση με τον RBF ή Gaussian πυρήνα, όπου θα πρέπει να γίνει σωστή ρύθμιση όλων των παραμέτρων, κάτι που είναι δύσκολο και χρονοβόρο [22].

Η **παράμετρος nu**, εκφράζει το μέγιστο ποσοστό από το σύνολο παραδειγμάτων εκπαίδευσης, που μπορεί να θεωρηθεί ως ανωμαλία [22]. Για παράδειγμα, η τιμή 0.05 υποδεικνύει ότι το πολύ 5% των παραδειγμάτων θα θεωρηθούν ανωμαλία (*outliers*). Όπως θα δούμε και παρακάτω, το σύνολο παραδειγμάτων εκπαίδευσης που χρησιμοποιούμε είναι άριστης ποιότητας, για αυτόν τον λόγο αποφασίσαμε να θέσουμε την παράμετρο nu με μια αρκετά μικρή τιμή, την 0.01.

Εξαγωγή χαρακτηριστικών (*feature extraction*)

- Local Binary Pattern

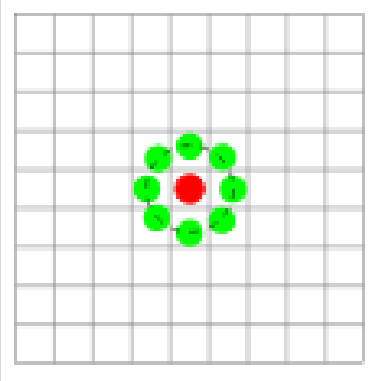
  Ο εξής αλγόριθμος εξαγωγής χαρακτηριστικών αποτελεί την βάση για τους επόμενους δύο αλγορίθμους που θα παρουσιάσουμε παρακάτω. Ο LBP [9] είναι αρκετά διαδεδομένος και γενικά αποτελεσματικός κατά την εφαρμογή του σε ψηφιακές εικόνες.

- Δημιουργούμε ένα ιστόγραμμα με τιμές 0 έως 255
- Για κάθε pixel της εικόνας εφαρμόζουμε τα ακόλουθα:

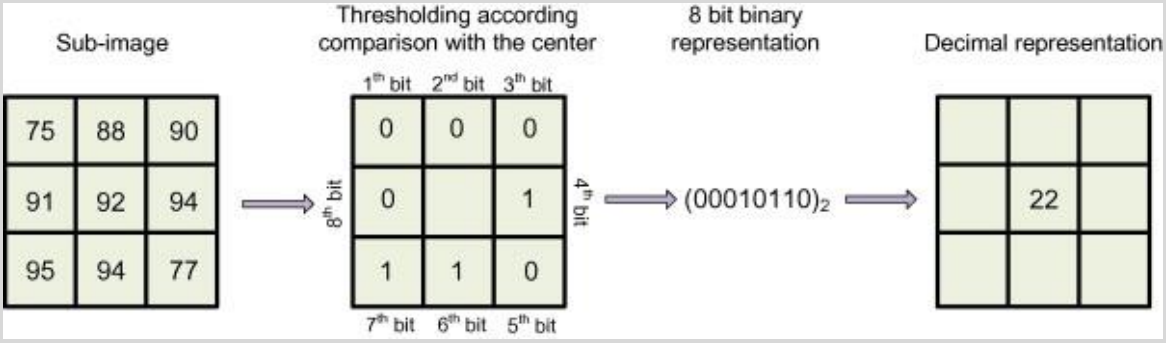
- Συγκρίνουμε τις τιμές χρώματος μεταξύ του εκάστοτε pixel (κεντρικό pixel) και των 8 άμεσα γειτονικών του pixel.
- Όσα γειτονικά pixel έχουνε χρωματική τιμή μεγαλύτερη ή ίση αυτής του κεντρικού pixel, τοποθετούμε σε αυτά την τιμή 1, αλλιώς την τιμή 0.
- Θέτοντας καθολικά, ένα οποιοδήποτε γειτονικό pixel ως το πρώτο και μια οποιαδήποτε φορά, παράγουμε έναν 8-ψήφιο αριθμό στο δυαδικό σύστημα.
- Τον μετατρέπουμε σε δεκαδικό.
- Αυξάνουμε κατά ένα την συχνότητα του δεκαδικού αριθμού στο ιστόγραμμά μας.

Αφού έχει ολοκληρωθεί η εφαρμογή του αλγορίθμου LBP στην εικόνα μας, μετατρέπουμε το ιστόγραμμά μας σε ένα διάνυσμα 256 διαστάσεων, αποτελούμενο μόνο από τις τιμές συχνότητας. Αυτό είναι και το διάνυσμα εισόδου εκπαίδευσης του SVM μοντέλου μας και αναπαριστά μια περιοχή εικόνας.

Εικόνα 2.10 - Κεντρικό και γειτονικά pixels



Εικόνα 2.11 – Εύρεση της LBP τιμής για ένα pixel



- Uniform Local Binary Pattern (ULBP)

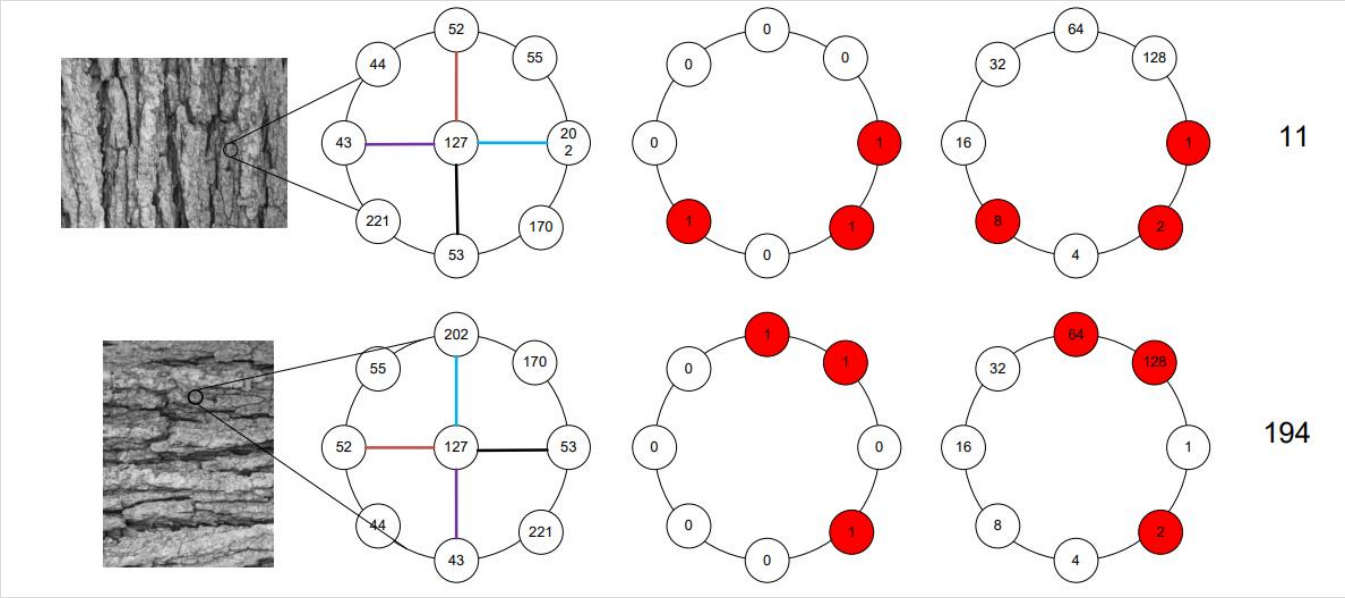
Ο ULBP αποτελεί επέκταση του LBP. Έχει παρατηρηθεί [15] πως ορισμένοι από τους 256 πιθανούς αριθμούς που μπορούμε να έχουμε ως αποτέλεσμα από την εφαρμογή του LBP, εμφανίζονται συχνότερα σε εικόνες υφής συγκριτικά με κάποιος άλλους αριθμούς. Οι αριθμοί με υψηλή συχνότητα εμφάνισης λέγονται ομοιόμορφοι (*uniform*). Ένας δυαδικός αριθμός θεωρείται ομοιόμορφος εάν ο μέγιστος αριθμός εναλλαγών των ψηφίων του είναι δύο. Για παράδειγμα ο αριθμός 00010000 (2 εναλλαγές) είναι ομοιόμορφος ενώ ο αριθμός 01010100 (6 εναλλαγές) δεν είναι. Οι ομοιόμορφοι αριθμοί μεταξύ του 0 και του 255 είναι 58. Έτσι, οι διαστάσεις του διανύσματος χαρακτηριστικών μας, μειώνονται από 256 στις 59, οι οποίες πλέον θα αναπαριστούν μόνο τους ομοιόμορφους αριθμούς συν μία ακόμα θέση για όλους τους μη-ομοιόμορφους. Η τεχνική αυτή, μειώνει σημαντικά το υπολογιστικό κόστος, αυξάνει την ταχύτητα εκπαίδευσης και μειώνει την κατανάλωση μνήμης του υπολογιστή.

- Uniform Rotated Local Binary Pattern (URLBP)

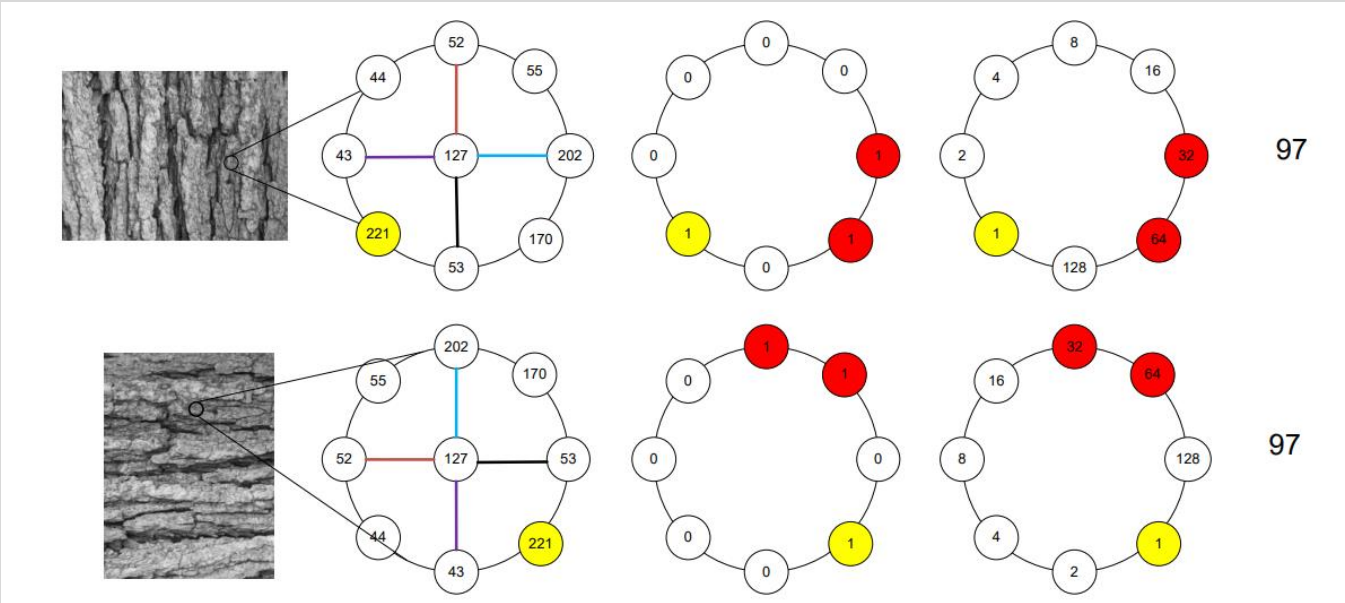
Ο URLBP αποτελεί προέκταση του ULBP [15]. Σκοπός της επέκτασης αυτής, είναι η εξαγωγή των χαρακτηριστικών μέσω του ULBP να είναι αμετάβλητη ως προς την περιστροφή της εικόνας. Οι διαστάσεις του διανύσματος χαρακτηριστικών που παράγει αυτός ο αλγόριθμος είναι ίσες με αυτές που παράγει ο ULBP. Ο URLBP αποτελείται από τα εξής βήματα:

- Δημιουργούμε ένα ιστόγραμμα με 59 θέσεις
- Για κάθε pixel της εικόνας εφαρμόζουμε τα ακόλουθα:
- Συγκρίνουμε τις τιμές χρώματος μεταξύ του εκάστοτε pixel (κεντρικό pixel) και των 8 άμεσα γειτονικών του.
- Βρίσκουμε το γειτονικό pixel με την μεγαλύτερη τιμή διαφοράς
- Ορίζουμε το pixel αυτό ως το πρωταρχικό
- Για όσα γειτονικά pixel έχουνε χρωματική τιμή μεγαλύτερη ή ίση αυτής του κεντρικού pixel, τοποθετούμε την τιμή 1, αλλιώς την τιμή 0.
- Ξεκινώντας από το πρωταρχικό γειτονικό pixel και έπειτα δεξιόστροφα, κατασκευάζουμε τον δυαδικό αριθμό μας.
- Τον μετατρέπουμε σε δεκαδικό.
- Αυξάνουμε την συχνότητα του δεκαδικού αριθμού στο ιστόγραμμά μας.

Εικόνα 2.12 - Παράδειγμα εφαρμογής του ULBP



Εικόνα 2.13 - Παράδειγμα εφαρμογής του URLBP



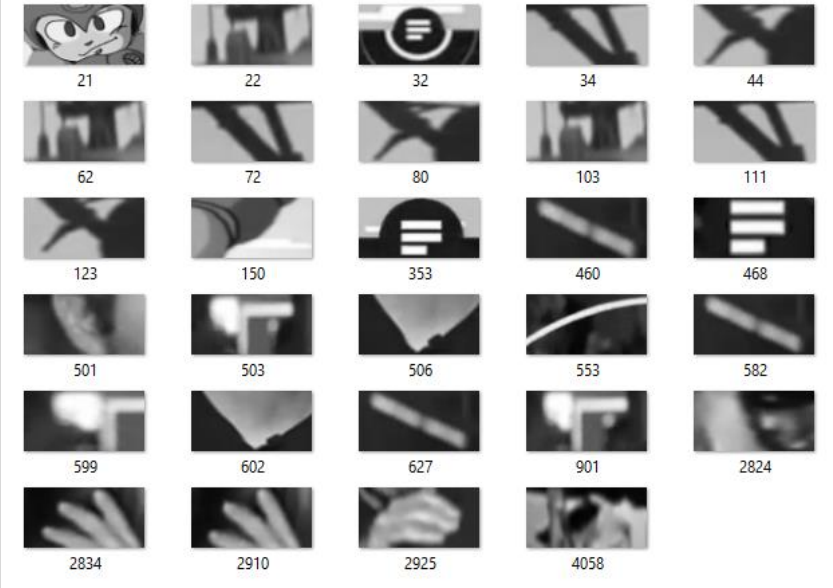
Σύνολα παραδειγμάτων (*datasets*)

Τα παραδείγματα εκπαίδευσης (*training dataset*) αποτελούνται μόνο από εικόνες «κειμένου» και είναι μέρος του συνόλου δεδομένων MJSynth [17]. Δημιουργήσαμε συνολικά έξι σύνολα παραδειγμάτων εκπαίδευσης με διαφορετικό μέγεθος το καθένα. Τα παραδείγματα ελέγχου (*testing dataset*) αποτελούνται από εικόνες «κειμένου» και «μη-κειμένου». Δημιουργήσαμε συνολικά τέσσερα σύνολα παραδειγμάτων ελέγχου, με 29 εικόνες το καθένα, όπου τα δύο σύνολα περιλαμβάνουν εικόνες «με κείμενο», ενώ τα άλλα δύο εικόνες «χωρίς κείμενο». Ένα ποσοστό των εικόνων ελέγχου είναι χειροκίνητα δημιουργημένες, ενώ οι υπόλοιπες είναι μέρος των συνόλων δεδομένων MJSynth και MSRA-TD500 [16]. Όλες οι εικόνες εκπαίδευσης και ελέγχου, ανήκουν στην κλίμακα του γκρίζου, με ύψος και πλάτος 50px και 100px αντίστοιχα.

Εκπαίδευση και έλεγχος μοντέλου (*model training*)

Έχοντας μια εικόνα ύψους 50px και πλάτους 100px, την διαιρούμε σε 50 υπο-εικόνες με ύψος και πλάτος 10px. Έτσι, έχουμε πλέον 50 υπο-εικόνες. Σε κάθε μία από αυτές, εφαρμόζουμε μία από τις μεθόδους εξαγωγής χαρακτηριστικών, την ULBP ή την URLBP. Κάθε υπο-εικόνα οπότε, αναπαρίσταται πλέον από ένα διάνυσμα 59 διαστάσεων. Αφού κανονικοποιήσουμε το διάνυσμα αυτό στο διάστημα τιμών 0-1 με την μέθοδο του ελαχίστου και μεγίστου [11], είμαστε έτοιμοι να το τροφοδοτήσουμε στο μοντέλο μας είτε για εκπαίδευση είτε για έλεγχο. Στην περίπτωση του ελέγχου, το μοντέλο θα πραγματοποιήσει την λεγόμενη πρόβλεψη και θα μας επιστρέψει για κάθε διάνυσμα, την τιμή 1 ή -1 εάν αυτό ανήκει στην θετική κατηγορία (περιέχει κείμενο) ή στην αρνητική (δεν περιέχει κείμενο) αντίστοιχα. Σε κάθε αρχική εικόνα επομένως, θα πρέπει να αντιστοιχεί ένας αριθμός θετικών και αρνητικών προβλέψεων με άθροισμα 50. Ο αλγόριθμος εξαγωγής χαρακτηριστικών που επιλέγεται για την εκπαίδευση, θα πρέπει να είναι ίδιος με αυτόν που επιλέγετε και για τον έλεγχο. Παρακάτω παρουσιάζονται τα σύνολα παραδειγμάτων ελέγχου που χρησιμοποιήθηκαν:

Εικόνα 2.14 – χωρίς_κείμενο_χειροκίνητα_δημιουργημένες



Εικόνα 2.15 - χωρίς_κείμενο_MSRA-TD500

Εικόνα 2.16 - κείμενο_χειροκίνητα_δημιουργημένες

Εικόνα 2.17 - κείμενο_MJSynth

Πίνακας 2.1 - Αριθμοί αρνητικών προβλέψεων (μη κείμενο) διάφορων μοντέλων ενάντια στα σύνολο ελέγχου

	«μη κειμένου» χειροκίνητα δημιουργημένες	«μη κειμένου» MSRA-TD500	«κειμένου» χειροκίνητα δημιουργημένες	«κειμένου» MJSynth
1000.model	492	168	56	120
10000.model	452	166	50	115
1000r.model	310	263	50	113
1500.model	474	172	54	115
2000.model	447	165	47	113
2000r.model	258	249	42	109
2500.model	455	169	51	114
2500r.model	275	253	46	112
3000.model	421	159	47	111
3000r.model	248	230	39	109

Σύμφωνα με την μεθοδολογία που παρουσιάσαμε προηγουμένως, η κάθε εικόνα θα διαιρεθεί σε 50 υπο-εικόνες. Συνολικά κάθε σύνολο ελέγχου θα έχει επομένως 29 * 50 = 1450 προβλέψεις. Ο Πίνακας 2.1 συγκρίνει την απόδοση του κάθε μοντέλου που εκπαιδεύσαμε ενάντια στα σύνολα ελέγχου. Η πρώτη στήλη περιέχει τα ονόματα των μοντέλων, με το αριθμητικό πρόθεμα να υποδηλώνει τον αριθμό των εικόνων που χρησιμοποιήθηκαν στην εκπαίδευση, ενώ ή ύπαρξη του γράμματος ‘r’ σημαίνει ότι χρησιμοποιήθηκε ο URLBP αλγόριθμος εξαγωγής χαρακτηριστικών, ειδάλλως χρησιμοποιήθηκε ο ULBP. Οι αριθμοί του πίνακα υποδεικνύουν πόσες υπο-εικόνες από τις 1450 του κάθε συνόλου ελέγχου, προβλέφθηκαν ως αρνητικές, δηλαδή ότι δεν περιέχουν κείμενο. Πάντοτε σε μια εικόνα, θα υπάρχουν χαρακτηριστικά που παραπέμπουν σε αυτά του κειμένου, για αυτό τον λόγο δεν περιμένουμε όλες οι υπο-εικόνες να κατηγοριοποιηθούν ως «κείμενο» ή «μη κείμενο» αλλά να δούμε μια ουσιαστική ποσοτική διαφορά μεταξύ των δύο περιπτώσεων κατανομών. Πράγματι, άμα παρατηρήσουμε τις κατανομές των αριθμών του πίνακα, θα δούμε πως για τα σύνολα παραδειγμάτων «μη κειμένου», οι υπο-εικόνες που είχαν αρνητική πρόβλεψη, δηλαδή ότι δεν περιέχουν κείμενο, είναι πολύ περισσότερες σε σύγκριση με αυτές των συνόλων παραδειγμάτων «κειμένου». **Έτσι, μπορούμε να θέσουμε έναν αριθμό αρνητικών προβλέψεων ως τιμή κατώφλι, η οποία θα είναι αυτή που θα καθορίζει εάν μια υπο-εικόνα περιέχει κείμενο ή όχι.** Διαισθητικά καταλήξαμε πως όταν ο αριθμός των υπο-εικόνων που κατηγοριοποιήθηκαν ως «μη κείμενο» είναι μεγαλύτερος από το 10% των συνολικών υπο-εικόνων, τότε η εικόνα αυτή δεν περιέχει κείμενο. Στην εφαρμογή μας, έχουμε επιλέξει να χρησιμοποιήσουμε το “10000.model” μοντέλο.

2.9 ΦΙΛΤΡΑΡΙΣΜΑ ΠΕΡΙΟΧΩΝ ΚΕΙΜΕΝΟΥ

Για να προχωρήσουμε στον τελικό προσδιορισμό των υποψήφιων περιοχών κειμένου που παρουσιάζει η Εικόνα 2.9, βρίσκουμε τα **συνεκτικά χωρία** αυτής. Τα συνεκτικά αυτά χωρία, αντιπροσωπεύουν όλες τις υποψήφιες περιοχές κειμένου της αρχικής μας εικόνας. Σκοπός μας είναι τώρα, να φιλτράρουμε τις περιοχές αυτές και να κρατήσουμε στο τέλος μονάχα αυτές που πραγματικά περιέχουν κείμενο.

Για να το πετύχουμε αυτό, εφαρμόζουμε την παρακάτω μεθοδολογία φιλτραρίσματος η οποία απορρίπτει τα χωρία τα οποία έχουν μικρές πιθανότητες να αναπαριστούν κάποια μορφή κειμένου. Να σημειώσουμε εδώ, ότι, βάσει του μέγιστου ύψους και πλάτους κάθε χωρίου, έχουμε παράξει ένα παραλληλόγραμμο το οποίο περιβάλλει το χωρίο. Παρακάτω, θεωρούμε το παραλληλόγραμμο αυτό ως το χωρίο μας. Η διαδικασία του φιλτραρίσματος χωρίζεται σε τρία στάδια. Όσα χωρία περάσουν το πρώτο στάδιο, φιλτράρονται από το δεύτερο και αυτά που θα περάσουν από το δεύτερο φιλτράρονται από το τρίτο. **Τα τελικά χωρία που θα απομείνουν είναι οι τελικές και οριστικές περιοχές κειμένου της εικόνας μας.** Ο αλγόριθμός φιλτραρίσματος αποτελείται από τα εξής στάδια:

1) <u>Βάσει των διαστάσεων των συνεκτικών χωρίων:</u>

   Ελέγχουμε και απορρίπτουμε σε περίπτωση αληθείας:

$$\frac{\Upsilon}{\Pi} > \mathrm{k}$$

   όπου:

   Υ : το μέγιστο ύψος του συνεκτικού χωρίου

   Π : το μέγιστο πλάτος του συνεκτικού χωρίου

   k: κατώφλι (=1.0 εμπειρική τιμή)

   Θεωρούμε πως για να υπάρξει ευανάγνωστο κείμενο θα πρέπει η αντίστοιχη περιοχή κειμένου να έχει μεγαλύτερο πλάτος από ύψος.

2) <u>Βάσει του εμβαδού του συνεκτικού χωρίου:</u>

   Ελέγχουμε και απορρίπτουμε σε περίπτωση αληθείας:

$$E < \Upsilon * \Pi * \kappa$$

όπου:

Ε: το εμβαδόν του χωρίου

Υ: το ύψος της εικόνας

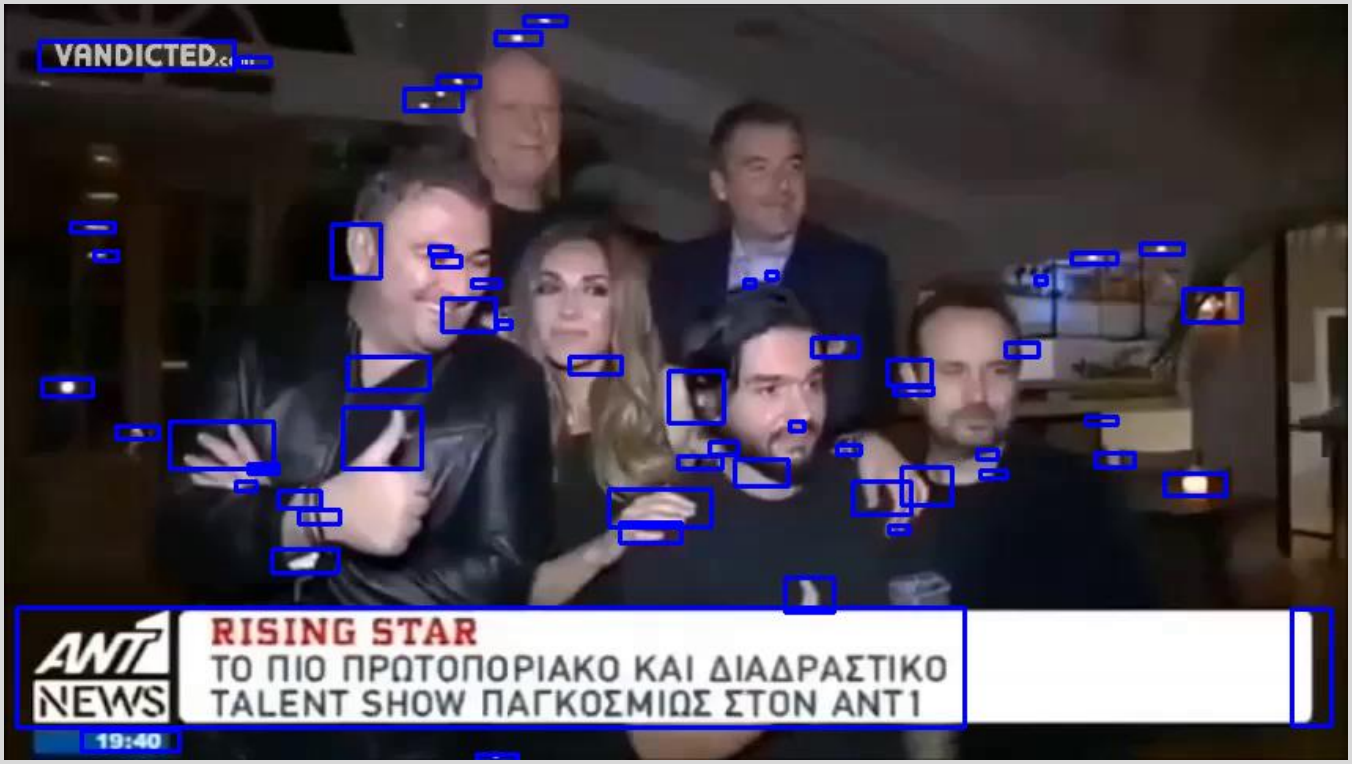
Π: το πλάτος της εικόνας

κ: σταθερά (=0.002 εμπειρική τιμή)

3) Βάσει της πρόβλεψης του SVM μοντέλου *(βλ. Ενότητα 2.8)*

Στην συνέχεια, παρουσιάζουμε την μεθοδολογία φιλτραρίσματος σε μορφή εικόνων, οι οποίες έχουνε παραχθεί από το λογισμικό που αναπτύχθηκε:

Εικόνα 2.18 - Όλες οι υποψήφιες περιοχές κειμένου – συνεκτικά χωρία από την Εικόνα 2.9



Εικόνα 2.19 - 1ο στάδιο φιλτραρίσματος – αφαίρεση περιοχών με μεγάλο ύψος

Εικόνα 2.20 - 2ο στάδιο φιλτραρίσματος – αφαίρεση περιοχών με μικρό εμβαδόν

Εικόνα 2.21 - 3ο στάδιο φιλτραρίσματος (τελικές περιοχές κειμένου) – αφαίρεση περιοχών σύμφωνα με την πρόβλεψη του SVM μοντέλου

2.10 ΕΞΑΓΩΓΗ ΚΕΙΜΕΝΟΥ

Έχοντας εντοπίσει τα χωρία της αρχικής εικόνας που περιέχουν κείμενο, θα πρέπει σε αυτό το σημείο να εφαρμόσουμε ορισμένα βήματα προεπεξεργασίας σε κάθε χωρίο, τα οποία θα μας οδηγήσουν στην σωστή και επιτυχημένη εξαγωγή του κειμένου μέσω της χρήσης Οπτικής Αναγνώρισης Χαρακτήρων (*OCR*) [19]. Αφού έχουμε αποκόψει τις τελικές περιοχές κειμένου, εφαρμόζουμε τα παρακάτω βήματα **προεπεξεργασίας** [18] σε κάθε μια από αυτές.

Τα βήματα προεπεξεργασίας είναι τα ακόλουθα:

- Αποκοπή εικόνας περιοχής κειμένου
- Μετατροπή σε κλίμακα του γκρίζου
- Εφαρμογή της μάσκας Unsharp Masking
- Δυαδική κατωφλίωση του Otsu

Η μάσκα Unsharp Masking ορίζεται ως εξής [25]:

$$U(x,y) = I(x,y) + k * g(x,y)$$

όπου,

U(x, y), η εικόνα αποτελέσματος της Unharp Masking

k, σταθερά κλιμάκωσης που συνήθως κυμαίνεται μεταξύ 0.2 – 0.7

$g(x,y) = f(x,y) - s(x,y)$

όπου, $s(x,y)$ η εξομάλυνση της εικόνας $f(x,y)$ *(βλ. Ενότητα 2.1)*

Ως παράδειγμα, ας δούμε την εφαρμογή της συνολικής προεπεξεργασίας σε ένα πλαίσιο ενός βίντεο που έχει επεξεργαστεί το λογισμικό μας:

Εικόνα 2.22 - Εντοπισμένες περιοχές κείμενου

Εικόνα 2.23 - Αποκομμένη περιοχή κειμένου

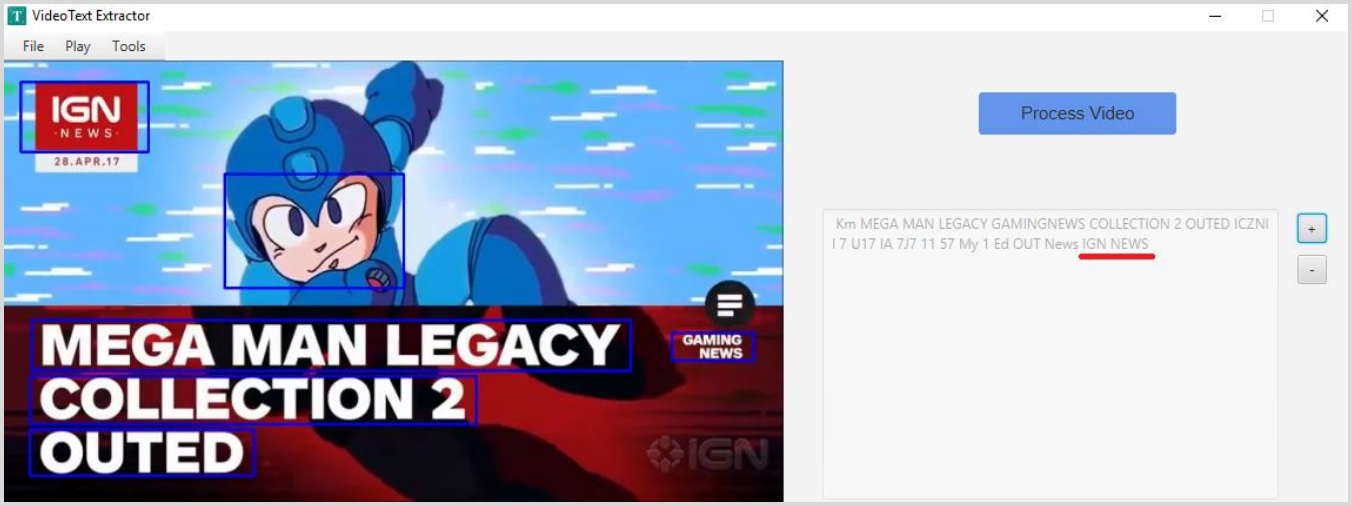
Εικόνα 2.24 - Μετατροπή σε κλίμακα του γκρίζου

Εικόνα 2.25 - Εφαρμογή της μάσκας Unsharp Masking

Εικόνα 2.26 - Δυαδική κατωφλίωση του Otsu

Μέσω της παραπάνω επεξεργασίας, έχουμε καταφέρει να απομονώσουμε ολοκληρωτικά το κείμενο της εικόνας από το παρασκήνιο, κάτι που είναι αυστηρή προϋπόθεση για να εφαρμοστεί πετυχημένα η τεχνική OCR. Σε αυτό το σημείο, η εικόνα μας βρίσκεται σε κατάλληλη μορφή ώστε να εφαρμόσουμε επιτυχημένα σε αυτήν την τεχνική OCR και να λάβουμε την αναπαράσταση του κειμένου σε μορφή Text.

Εικόνα 2.27 - Εφαρμογή της τεχνικής OCR – Εξαγωγή κειμένου



Η Εικόνα 2.27 αποτελεί απόσπασμα του λογισμικού που αναπτύχθηκε. Στην συνέχεια στο Κεφάλαιο 3, παρουσιάζεται αναλυτικά η ανάπτυξη της εφαρμογής, οι τεχνολογίες που χρησιμοποιήθηκαν, καθώς και αποσπάσματα κώδικα.

3 ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΗ ΑΝΑΛΥΣΗ

3.1 ΠΕΡΙΒΑΛΛΟΝ ΑΝΑΠΤΥΞΗΣ

Δεδομένου ότι η εφαρμογή μας θα πρέπει να παρέχει ένα οπτικοποιημένο περιβάλλον χρήσης, θα πρέπει να αναζητήσουμε τις αντίστοιχες τεχνολογίες οι οποίες μπορούν να μας προσφέρουν αυτήν την λειτουργία. Παράλληλα, θα χρειαστεί να συνδυάσουμε τις τεχνολογίες που επιλέξαμε με μια βιβλιοθήκη Μηχανικής Όρασης η οποία θα μας βοηθήσει στην υλοποίηση των μεθόδων και αλγορίθμων που αναλύσαμε προηγουμένως. Κριτήριο επιλογής των τεχνολογιών αλλά και στόχος της εφαρμογής μας είναι η ανεξαρτησία πλατφόρμας, η επεκτασιμότητα, η επαναχρησιμοποίηση του κώδικα, η σωστή σχεδίαση και αρχιτεκτονική αλλά και η αποδοτικότητά της.

- **Γλώσσα Προγραμματισμού**

Ως κύρια γλώσσα προγραμματισμού για την εφαρμογή μας επιλέξαμε την **Java**. Ένας από τους κύριους λόγους που επιλέξαμε την γλώσσα αυτή, είναι ότι η τελευταία προσφέρει μια πολύ δημοφιλής και ισχυρή βιβλιοθήκη ανάπτυξης οπτικοποιημένων εφαρμογών, όπως θα δούμε και στην συνέχεια. Για την γλώσσα αυτή παρέχεται αρκετά μεγάλη υποστήριξη στο Διαδίκτυο, πράγμα το οποίο αποδείχθηκε πολύ χρήσιμο τόσο για την σωστή σύνδεση της γλώσσας με τρίτες σημαντικές τεχνολογίες όσο και για την αποσφαλμάτωση της εφαρμογής κατά την πορεία ανάπτυξής της.

- **Ανάπτυξη Οπτικής Εφαρμογής**

Η Java παρέχει μια από τις πιο δυνατές και πιο δημοφιλής βιβλιοθήκες για ανάπτυξη οπτικών εφαρμογών. Μέχρι πρόσφατα, την θέση αυτήν κατείχε η βιβλιοθήκη Swing, αλλά τα τελευταία χρόνια έχει πάρει την θέση της μια ακόμα πιο προχωρημένη και καινούργια τεχνολογία η **JavaFX**, η οποία είναι και αυτή που επιλέξαμε. Η τελευταία είναι cross-platform και χρησιμοποιείται επίσης και για την ανάπτυξη smartphone εφαρμογών.

- **Μηχανική Όραση**

Αναμφισβήτητα η βιβλιοθήκη αυτή είναι το πιο σημαντικό στοιχείο της εφαρμογής. Πρόκειται για την πιο δημοφιλή, δωρεάν και ανοιχτού κώδικα βιβλιοθήκη, την **OpenCV**. Η τελευταία είναι γραμμένη στην C/C++ αλλά παρέχει υποστήριξη για αρκετές δημοφιλείς γλώσσες προγραμματισμού συμπεριλαμβανομένου και της Java. Η OpenCV παρέχει υλοποίηση κώδικα σχεδόν για οτιδήποτε

συσχετίζεται με την Μηχανική Όραση, όπου φυσικά συμπεριλαμβάνεται και η Ψηφιακή Επεξεργασία Εικόνας, που είναι και ο τομέας που μας ενδιαφέρει περισσότερο.

- **Μηχανική Μάθηση**

Για την εφαρμογή του SVM (Support Vector Machine) κατηγοριοποιητή, μελετήσαμε και χρησιμοποιήσαμε την βιβλιοθήκη μηχανικής μάθησης ανοιχτού κώδικα **LIBSVM**, η οποία έχει αναπτυχθεί από το πανεπιστήμιο National Taiwan University και είναι γραμμένη στην C++ αλλά παρέχει κι αυτή υποστήριξη για αρκετές γλώσσες προγραμματισμού όπως η Java.

- **GUI & Unit Testing**

Το Testing, δηλαδή ο έλεγχος της εφαρμογής είναι ζωτικό στοιχείο κάθε λογισμικού υψηλού επιπέδου. Μέσω αυτού μπορούμε και εντοπίζουμε σημαντικά προγραμματιστικά και λογικά λάθη της εφαρμογής τα οποία δύσκολα θα τα βρίσκαμε αλλιώς. Επίσης μπορούμε να δούμε πώς ανταποκρίνεται η εφαρμογή μας σε ακραία και αθέμιτα σενάρια και περιπτώσεις και τι έχουμε προβλέψει εμείς ως προγραμματιστές για να τα αντιμετωπίσουμε. Το Test Coverage, δηλαδή το ποσοστό του πηγαίου κώδικά μας που έχει ελεγχθεί από τα tests μας, παρέχει μια εικόνα για την σταθερότητα την εφαρμογής αλλά και για την σωστή και αναμενόμενη λειτουργία της. Χρησιμοποιήσαμε την βιβλιοθήκη **JUnit** για την συγγραφή των Unit Tests και την βιβλιοθήκη **TestFX** για τον έλεγχο του GUI.

- **Build Automation System**

Ένα ακόμα σημαντικό κομμάτι κάθε εφαρμογής είναι το build tool που χρησιμοποιεί. Σκοπός του είναι να χτίζει ολοκληρωμένα την εφαρμογή μας, βάσει των δικών μας ρυθμίσεων και να συλλέγει για εμάς τις online και local βιβλιοθήκες που χρειαζόμαστε για το project μας. Επιπλέον, μέσω του build tool που χρησιμοποιήσαμε, μπορεί να γίνει άμεσα και εύκολα build και run της εφαρμογής σε έναν τρίτο υπολογιστή. Τέλος, μέσω του ***javafx-gradle-plugin*** μπορέσαμε και δημιουργήσαμε εκτελέσιμα αρχεία της εφαρμογής που είναι φορητά και τρέχουν ανεξάρτητα σε Windows και Linux. Όλα τα παραπάνω υλοποιήθηκαν μέσω του **Gradle** ενός open-source build tool που είναι πολύ δημοφιλές ειδικά για την γλώσσα Java.

- **Version Control**

Αν και συνηθίζεται να χρησιμοποιείται σε ομαδικά projects, υιοθετήσαμε την τεχνολογία **Git** σε αυτό το ατομικό project για την διαχείριση του κώδικα αλλά και για την παραγωγή χρήσιμων διαγραμμάτων μέσω του GitHub τα οποία θα μελετήσουμε στην Ενότητα 3.4.

- **IDE**

Το ολοκληρωμένο περιβάλλον ανάπτυξης λογισμικού που χρησιμοποιήσαμε ήτανε το **Intelij IDEA** από την εταιρία JetBrains, το οποίο είναι ένα από τα πιο ευφυή και διαδεδομένα λογισμικά που πετυχαίνουν τεράστια αύξηση της παραγωγικότητας του προγραμματιστή.

- **Λοιπές Βιβλιοθήκες**

Πέραν από τις βιβλιοθήκες που αναφέρθηκαν, χρησιμοποιήθηκαν και άλλες για την επίτευξη διάφορων σκοπών της εφαρμογής μας, όπως: **Google Guava**, **CERN Colt** και **Apache Commons.** Επίσης χρησιμοποιήσαμε την βιβλιοθήκη **JavaCPP-Tesseract** και την **languagetool** για την εφαρμογή της τεχνικής OCR, **CSS** για την μορφοποίηση του GUI και τέλος Java 8 χαρακτηριστικά όπως **lambda expressions,** νήματα (**threads**) και άλλα.

- **Αρχιτεκτονική και Σχεδίαση**

Τέλος, πέρα από τις βιβλιοθήκες και τεχνολογίες που αναφέραμε, χρησιμοποιήσαμε την **MVC** (Model View Controller) αρχιτεκτονική σχεδίασης. Δεν υπάρχει κάποιο συγκεκριμένο και αυστηρό πρότυπο υλοποίησης αυτής, καθώς αποτελεί μια γενικευμένη ιδέα, οπότε προσαρμόσαμε αυτή την ιδέα στην JavaFX εφαρμογή μας.

3.2 ΥΛΟΠΟΙΗΣΗ ΜΕΘΟΔΟΛΟΓΙΑΣ

Σε αυτήν την ενότητα, θα γίνει η συνοπτική παρουσίαση του κώδικα που χρησιμοποιήθηκε για την υλοποίηση της μεθοδολογίας που παρουσιάσαμε στο Κεφάλαιο 2. Όπως έχουμε ήδη αναφέρει, το πρώτο βήμα που θα πρέπει να εφαρμόσουμε είναι να αποσπάσουμε πλαίσια από το βίντεο μας, με μια ορισμένη συχνότητα (*ένα ανά πέντε*) από το βίντεο μας, ώστε να τα επεξεργαστούμε.

Όπως είναι λογικό να φανταστεί κανείς, είναι αρκετά πιθανόν να λάβουμε πολλαπλά όμοια αποτελέσματα εξαγωγής κειμένου ή αποτελέσματα που αφορούν «μισές» λέξεις και κείμενα που δεν έχουν προφτάσει να ολοκληρωθούν στο εκάστοτε πλαίσιο. Μπορούμε να λύσουμε αυτό το πρόβλημα καθαρά προγραμματιστικά ελέγχοντας την συσχέτιση των αποτελεσμάτων όπως θα δούμε και στην συνέχεια.

Κώδικας 3.1 - Επιλογή πλαισίων

```
Mat initialFrame = new Mat();
VideoCapture cap = new VideoCapture("video.mp4");
boolean opened = cap.read(initialFrame);
```

Δημιουργούμε ένα αντικείμενο με όνομα `cap` τύπου `VideoCapture` και το αρχικοποιούμε με το ψηφιακό πολυμεσικό αρχείο που επιθυμούμε να επεξεργαστούμε. Η κλάση που χρησιμοποιούμε ορίζεται στην OpenCV. Καλώντας τη μέθοδο `read` διαβάζουμε το πρώτο πλαίσιο από το βίντεο και το αποθηκεύουμε στο αντικείμενο `initialFrame` τύπου `Mat` το οποίο έχουμε εισάγει ως παράμετρο. Η κλάση `Mat` είναι από τις βασικότερες της OpenCV καθώς είναι αυτή που αντιπροσωπεύει κάθε μορφή εικόνας. Η μέθοδος επιστρέφει μια `true` ή `false` τιμή, υποδεικνύοντας άμα το πλαίσιο διαβάστηκε με επιτυχία ή όχι. Μια `false` τιμή σημαίνει ότι δεν υπάρχει άλλο πλαίσιο για να διαβαστεί, δηλαδή έχουνε διαβαστεί όλα τα πλαίσια του βίντεο ή απλά δεν υπάρχουν πλαίσια για να διαβαστούν.

Κώδικας 3.2 - Εφαρμογή του φίλτρου Gaussian Blur

```
Imgproc.GaussianBlur(initialFrame, gaussianBlurredImage,
                        new Size(15.0,15.0),0.0,0.0);
```

Η μέθοδος `GaussianBlur` ορίζεται στην OpenCV και οι παράμετροί της περιγράφονται ως εξής:

```
src - αρχική εικόνα τύπου Mat
dst - εικόνα αποτελέσματος με ίδιες διαστάσεις με την αρχική
ksize - οι διαστάσεις του Gaussian πυρήνα τύπου Size
sigmaX - Η τυπική απόκλιση στην κατεύθυνση X, τύπου double
sigmaY - Η τυπική απόκλιση στην κατεύθυνση Y, τύπου double
```

Κώδικας 3.3 - Μετατροπή από RGB σε κλίμακα του γκρίζου

```
Imgproc.cvtColor(gaussianBlurredImage,grayImage,
                                Imgproc.COLOR_RGB2GRAY);
```

Η μέθοδος `cvtColor` ορίζεται στην OpenCV. Στην τρίτη παράμετρο δηλώνουμε πως πρόκειται για μετατροπή χρωματικού χώρου από RGB σε κλίμακα του γκρίζου.

Κώδικας 3.4 - – Εφαρμογή του φίλτρου Laplace

```
Imgproc.Laplacian(grayImage,
                  laplacianImage, CvType.CV_16S, 3);
```

Η μέθοδος `Laplacian` ορίζεται στην OpenCV και εφαρμόζει το φίλτρο Laplace στην εικόνα κλίμακας του γκρίζου, την οποία δίνουμε ως πρώτη παράμετρο και αποθηκεύει το αποτέλεσμα στη δεύτερη παράμετρο, όπου δίνουμε ένα καινούργιο `Mat` αντικείμενο. Οι παράμετροί της ορίζονται ως εξής:

`GrayImage` – αρχική εικόνα τύπου Mat
`LaplacianImage` – εικόνα αποτελέσματος με ίδιες διαστάσεις με την αρχική
`CvType.CV_16SC1` – Ο τύπος της εικόνας αποτελέσματος (βλ. Εικόνα 3.1 - `Τύποι Mat της OpenCV`)
`3` – Οι διαστάσεις του τετραγωνικού Laplacian πυρήνα. Θα πρέπει να είναι θετικός και περιττός αριθμός. Μεγαλύτερος αριθμός οδηγεί σε εντονότερο αποτέλεσμα.

Να σημειώσουμε εδώ ότι ο τύπος την εικόνας που αναφέρουμε στην τρίτη παράμετρο πηγάζει από τους ειδικούς τύπους `Mat` της OpenCV. Για να επιλέξουμε τον κατάλληλο τύπο από την κλάση `CvType`, θα πρέπει πρώτα να συμβουλευτούμε τον παρακάτω πίνακα, που περιγράφει του τύπους Mat της OpenCV [23]:

Εικόνα 3.1 - Τύποι Mat της OpenCV

13

A Mapping of Type to Numbers in OpenCV

	C1	C2	C3	C4
CV_8U	0	8	16	24
CV_8S	1	9	17	25
CV_16U	2	10	18	26
CV_16S	3	11	19	27
CV_32S	4	12	20	28
CV_32F	5	13	21	29
CV_64F	6	14	22	30

Unsigned 8bits uchar 0~255
IplImage: IPL_DEPTH_8U
Mat: CV_8UC1, CV_8UC2, CV_8UC3, CV_8UC4

Signed 8bits char -128~127
IplImage: IPL_DEPTH_8S
Mat: CV_8SC1, CV_8SC2, CV_8SC3, CV_8SC4

Unsigned 16bits ushort 0~65535
IplImage: IPL_DEPTH_16U
Mat: CV_16UC1, CV_16UC2, CV_16UC3, CV_16UC4

Signed 16bits short -32768~32767
IplImage: IPL_DEPTH_16S
Mat: CV_16SC1, CV_16SC2, CV_16SC3, CV_16SC4

Signed 32bits int -2147483648~2147483647
IplImage: IPL_DEPTH_32S
Mat: CV_32SC1, CV_32SC2, CV_32SC3, CV_32SC4

Float 32bits float -1.18*10-38~3.40*10-38
IplImage: IPL_DEPTH_32F
Mat: CV_32FC1, CV_32FC2, CV_32FC3, CV_32FC4

Double 64bits double
Mat: CV_64FC1, CV_64FC2, CV_64FC3, CV_64FC4

Unsigned 1bit bool
IplImage: IPL_DEPTH_1U

Στο σημείο αυτό θέλουμε να μετατρέψουμε προσωρινά το αντικείμενο τύπου `Mat` που περιέχει την τωρινή κατάσταση της εικόνας μας σε έναν δισδιάστατο πίνακα τύπου `double`. Ο λόγος που κάνουμε την μετατροπή αυτή είναι ότι θα μας βοηθήσει να αποκτήσουμε πρόσβαση σε όλες τις τιμές χρώματος του κάθε pixel εύκολα ώστε να υπολογίσουμε τον MGD συντελεστή στην συνέχεια.

Κώδικας 3.5 - Μετατροπή από Mat σε Array

```
laplacianArray = PixelProcessor.matToArray(laplacianImage);
```

Εκτελούμε την μέθοδο `matToArray` η οποία θα μετατρέψει την `Mat` εικόνα που δίνουμε ως παράμετρο σε έναν δισδιάστατο πίνακα τύπου `double`. Η μέθοδος `matToArray` ανήκει στην κλάση `PixelProcessor` και έχει συγγραφεί από εμάς.

Κώδικας 3.6 - Υπολογισμός του τελεστή MGD

```
mgdArray = PixelProcessor.calculateMGD(laplacianArray));
```

Εκτελούμε την μέθοδο `calculateMGD` η οποία ανήκει στην κλάση `PixelProcessor` και έχει συγγραφεί από εμάς. Η παράμετρος που δέχεται είναι ο πίνακας με τις τιμές των pixel της εικόνας που δημιουργήσαμε προηγουμένως. Θα υπολογίσει για κάθε τιμή τον MGD συντελεστή και θα επιστρέψει έναν καινούργιο πίνακα με τις καινούργιες αντίστοιχες τιμές.

Κώδικας 3.7 - Μετατροπή από Array σε Mat

```
mgdImage = PixelProcessor.arrayToMat(mgdArray);
```

Εφόσον έχουνε ολοκληρωθεί οι υπολογισμοί που έπρεπε να γίνουν εκμεταλλευόμενοι τη δομή του πίνακα, τώρα μπορούμε να μετατρέψουμε τον πίνακά μας ξανά σε αντικείμενο τύπου `Mat`. Για να γίνει αυτό, καλούμε τη μέθοδο `arrayToMat` της κλάσης `PixelProcessor` η οποία έχει συγγραφεί από εμάς.

Κώδικας 3.8 – Απλή Δυαδική κατωφλίωση

```
Imgproc.threshold(mgdImage, binaryImage, 80,255,Imgproc.THRESH_BINARY);
```

Μετατρέπουμε την `mgdImage` εικόνα, σε δυαδική, χρησιμοποιώντας την μέθοδο απλής δυαδικής κατωφλίωσης, με τιμή κατωφλίου `80`. Η παράμετρος `255` δηλώνει ποια χρωματική τιμή θα λάβουν τα pixel της εικόνας που ξεπερνούν την τιμή κατωφλίου. Έτσι, τα pixel της εικόνας με τιμή `>80` θα χρωματιστούν με `255` (άσπρο) ενώ τα υπόλοιπα με την τιμή 0 (μαύρο),

Κώδικας 3.9 - Μορφολογική Διαστολή

```
Imgproc.dilate(binaryImage, dilatedImage,
                    Imgproc.getStructuringElement(
                                        Imgproc.MORPH_RECT, new Size(13.0,13.0)));
```

Χρησιμοποιώντας την μέθοδο `dilate` της OpenCV, εισάγουμε ως παραμέτρους τη δυαδική εικόνα στην οποία θέλουμε να εφαρμόσουμε την μορφολογική πράξη της διαστολής, την εικόνα όπου θα αποθηκευτεί το αποτέλεσμα, αλλά και το δομικό στοιχείο που θα χρησιμοποιήσουμε, στο οποίο έχουμε δώσει διαστάσεις 13 x 13 μορφής τετραγώνου. Για να έχουμε βέλτιστα αποτελέσματα, θα πρέπει οι διαστάσεις του δομικού στοιχείου να είναι ανάλογες των διαστάσεων του βίντεο. Για αυτό τον λόγο, η εφαρμογής μας επιλέγει αυτόματα τις διαστάσεις του δομικού στοιχείου για κάθε βίντεο, εξετάζοντας τις διαστάσεις του. Η άγκυρα βρίσκεται στο κέντρο του δομικού στοιχείο ως προεπιλογή και είναι αυτό που επιθυμούμε. Το πρώτο βήμα του φιλτραρίσματος είναι να βρούμε τα συνεκτικά χωρία από την εικόνα που εφαρμόσαμε την μορφολογική διαστολή:

Κώδικας 3.10 - Εύρεση συνεκτικών χωρίων

```
Mat labels = new Mat();
Mat stats = new Mat();
Mat centroids = new Mat();
int numberOfLabels =
Imgproc.connectedComponentsWithStats(dilatedImage,labels,stats,
                                        centroids,8, CvType.CV_32S);
```

Η μέθοδος `connectedComponentsWithStats` της OpenCV δέχεται ως παραμέτρους τέσσερα `Mat` αντικείμενα, έναν ακέραιο αριθμό και τον τύπο της εικόνας. Για τον τύπο της εικόνας συμβουλευόμαστε και πάλι την Εικόνα 3.1. Το αντικείμενο `dilated` είναι αυτό του οποίου θέλουμε να βρούμε τα συνεκτικά χωρία. Στο αντικείμενο `labels` θα αποθηκευτούν οι ταμπέλες για κάθε χωρίο, όπου το παρασκήνιο έχει την ταμπέλα 0. Το αντικείμενο `stats`, που είναι και το πιο σημαντικό θα έχει διαστάσεις 5 x N, όπου N ο αριθμός των εντοπισμένων χωρίων. Στην παράμετρο centroids θα αποθηκευτούν πληροφορίες που αφορούν τα κέντρα των χωρίων. Ο ακέραιος αριθμός, το 8 δηλαδή, είναι το η συνδεσιμότητα και δηλώνει με πόσα γειτονικά pixels μπορεί να συνδεθεί το κάθε pixel ώστε να δημιουργηθεί πιθανό συνεκτικό χωρίο. Τέλος, στην τελευταία παράμετρο δίνουμε τον τύπο της εικόνας.

Πίνακας 3.1 - Δομή αντικειμένου stats

left	*top*	*width*	*height*	*area*
0	0	800	600	421697
100	100	101	101	10201
500	150	101	301	304401
350	246	10	10	36
255	325	151	151	17665

Κάθε γραμμή του αντικειμένου `stats` παριστάνει ένα συνεκτικό χωρίο, δίνοντας παράλληλα χρήσιμες πληροφορίες για αυτό όπως τις συντεταγμένες του, το μέγεθός του και το εμβαδόν του. Δεδομένου ότι θέλουμε να ξέρουμε τις περιοχές της εικόνας που αντιστοιχούν στα χωρία, οι πληροφορίες αυτές είναι εξαιρετικά χρήσιμες.

Κώδικας 3.11 - Φιλτράρισμα συνεκτικών χωρίων – Εντοπισμός περιοχών κειμένου

```
List<Rect> textBlocks = new ArrayList<>();
// Label 0 is considered to be the background label, so we skip it
for (int i = 1; i < numberOfLabels; i++)
{
    Rect textBlock = new Rect(
                          new Point(stats.get(i,0)[0],stats.get(i,1)[0]),
                          new Size(stats.get(i,2)[0],
                          stats.get(i,3)[0]));
    Mat crop = new Mat(VideoProcessor.getInput(),textBlock);
    // FILTER 1
    if ( Double.compare(textBlock.width / textBlock.height, 1.0) >= 0) {
        // FILTER 2
        if (Double.compare(stats.get(i,4)[0],
            dilated.height() * dilated.width() * 0.002 ) > 0){
                Imgproc.cvtColor(crop, crop, Imgproc.COLOR_RGB2GRAY, 0);
                Imgproc.resize(crop, crop,
                new Size(100,50), 4.0, 4.0,Imgproc.INTER_LINEAR);
                // FILTER 3
                if (SVM.blockContainsText(crop)){
                    textBlocks.add(textBlock);
                }
        }
    }
}
```

Στο παραπάνω παράδειγμα κώδικα βλέπουμε την υλοποίηση της μεθοδολογίας φιλτραρίσματος των υποψήφιων περιοχών κειμένου όπως αυτή παρουσιάστηκε στην Ενότητα 2.9. Χρησιμοποιώντας το αντικείμενο `stats` του προηγουμένου βήματος που περιέχει τα συνεκτικά χωρία, δημιουργούμε παραλληλόγραμμα σύμφωνα με τις συντεταγμένες και τα τις διαστάσεις κάθε χωρίου. Έπειτα, ως πρώτο

βήμα του φιλτραρίσματος, ελέγχουμε άμα το πλάτος του παραλληλόγραμμου είναι μεγαλύτερο του ύψους του. Θεωρούμε πως ένα τέτοιο χαρακτηριστικό είναι απαραίτητη προϋπόθεση για να υφίσταται ευανάγνωστο και έγκυρο κείμενο στην περιοχή του παραλληλόγραμμου. Για τα παραλληλόγραμμα που ικανοποιούν το παραπάνω κριτήριο, ελέγχουμε άμα το εμβαδόν τους είναι μεγαλύτερο από ένα κατώφλι μεγέθους. Πολλές φορές, ως αποτέλεσμα της μεθοδολογίας μας, τυχαίνει να παράγονται πολλές υποψήφιες περιοχές κειμένου με πολύ μικρό εμβαδόν. Συνήθως οι περιοχές αυτές δε φέρουν καμία πληροφορία και επομένως θα πρέπει να αφαιρεθούν. Τέλος, οι περιοχές που θα έχουν ικανοποιήσει και το κριτήριο του εμβαδού, μπορούν πλέον να περάσουν στο τρίτο και τελευταίο στάδιο του φιλτραρίσματος που είναι η πρόβλεψη του SVM κατηγοριοποιητή. Όπως εξηγήσαμε και στην Ενότητα 2.8, άμα το μοντέλο προβλέψει έναν αριθμό αρνητικών υπο-περιοχών μεγαλύτερο από ένα συγκεκριμένο κατώφλι, τότε η περιοχή αυτή της εικόνας θεωρούμε ότι δεν περιέχει κείμενο.

Χρησιμοποιώντας την μεθοδολογία του φιλτραρίσματος που παρουσιάσαμε παραπάνω, έχουμε καταφέρει και αποκόψει τα χωρία που δεν αναπαριστούν κείμενο. Έχοντας λοιπόν τα χωρία που περιέχουν κείμενο, μπορούμε να προχωρήσουμε στην εξαγωγή του κειμένου που αναπαριστούν. Αυτό θα πετύχουμε μέσω της γνωστής τεχνικής OCR, αλλά για να γίνει αυτή επιτυχημένα θα πρέπει πρώτα να προεπεξεργαστούμε τις περιοχές αυτές τις εικόνας.

Κώδικας 3.12 - Προεπεξεργασία OCR

```
// Convert to Grey scale
Imgproc.cvtColor(textBlock, src, Imgproc.COLOR_RGB2GRAY, 0);

// Unsharp Mark
Imgproc.GaussianBlur(src, dst, new Size(0, 0), 3);
Core.addWeighted(src, 1.5, dst, -0.5, 0, unsharp);

// Normalize image
Core.normalize(unsharp, src,0.0,1.0, Core.NORM_MINMAX);

// Otsu Binarization
Imgproc.threshold(unsharp,dst, 80,255,Imgproc.THRESH_BINARY + THRESH_OTSU);
```

Όλες οι μέθοδοι που βλέπουμε παραπάνω ανήκουν στην OpenCV. Για την εφαρμογή της Unsharp Mark θα πρέπει να συνδυάσουμε δύο μεθόδους. Η μέθοδος `addWeighted` θα προσθέσει τα βάρη των δύο εικόνων όπου η πρώτη είναι η εικόνα μας στην κλίμακα του γκρίζου και η δεύτερη είναι η εικόνα μετά την εφαρμογή του φίλτρου του Gauss. Ορίζουμε ως βάρη το 1.5 και -0.5 για την κάθε μια εικόνα αντίστοιχα και έτσι πετυχαίνουμε το αποτέλεσμα της Unsharp Mask. Έπειτα, αφού εφαρμόσουμε την δυαδική κατωφλίωση του Otsu, μπορούμε να προχωρήσουμε στην εφαρμογή της OCR τεχνικής.

Κώδικας 3.13 - Εξαγωγή κειμένου με OCR

```
BytePointer outText;

TessBaseAPI api = new TessBaseAPI();

// Initialize tesseract-applyOCR with English
api.Init("src/main/resources/", "ENG"))

// Open input image with leptonica library
PIX image = pixRead(matPath);
api.SetImage(image);

// Get OCR result
outText = api.GetUTF8Text();

String ocr_text = outText.getString();
System.out.println("OCR output:\n" + ocr_text);
```

Έχοντας εισάγει την βιβλιοθήκη JavaCPP-Tesseract, δηλώνουμε ότι η γλώσσα που επιθυμούμε να εξάγουμε είναι για παράδειγμα τα Αγγλικά, έπειτα διαβάζουμε την εικόνα/περιοχή κειμένου και την αποθηκεύουμε σε ένα αντικείμενο τύπου `PIX` και ύστερα και στο αντικείμενο `api`. Τέλος, καλώντας την μέθοδο `GetUTF8Text` λαμβάνουμε το τελικό αποτέλεσμα της εξαγωγής κειμένου σε μορφή `String`.

Έλεγχος αποτελέσματος

Βέβαια, όπως έχουμε προαναφέρει, η μεθοδολογία μας πολλές φορές πιθανόν να εντοπίσει την ίδια περιοχή κειμένου ή κάποια περιοχή που το κείμενό της δεν έχει προφτάσει ακόμη να ολοκληρωθεί. Η τεχνική OCR επιχειρεί πάντοτε να εξάγει το καλύτερο δυνατό αποτέλεσμα σύμφωνα με την εικόνα που έχουμε εισάγει. Αυτό σημαίνει, πως σε περίπτωση που εφαρμόσουμε την τεχνική σε μια εικόνα που δεν περιέχει κείμενο, τότε τα αποτελέσματα εξαγωγής θα είναι λέξεις και χαρακτήρες που δεν έχουν συνηθισμένη συνοχή. Εκμεταλλευόμενοι την ιδιότητα αυτή, εφαρμόσαμε ένα τελικό στάδιο φιλτραρίσματος, όπου ελέγχουμε την εγκυρότητα των OCR αποτελεσμάτων πριν τα παρουσιάσουμε στον χρήστη. Μέσω της εφαρμογής μπορεί να ενεργοποιηθεί ή να απενεργοποιηθεί οποιοσδήποτε από τους παρακάτω ελέγχους. Ο έλεγχος εγκυρότητας αυτός διακρίνεται σε τρία μέρη:

Κώδικας 3.14 - Αφαίρεση όμοιων λέξεων

```
String[] words = ocrOutput.getString().trim().split(" ");
for (String word : words){
    if (extractUniqueWords) {
        if (uniqueWords.contains(word)){
            continue;
        }
        uniqueWords.add(word);
        appendToTextArea(word);
    }
}
```

Με την τεχνική αυτή, αντιμετωπίζουμε τον πολλαπλό εντοπισμό όμοιων περιοχών κειμένου. Μπορεί να φανεί ιδιαίτερα χρήσιμη όταν μας ενδιαφέρει κυρίως η ύπαρξη μιας λέξης και όχι η συχνότητα εμφάνισής της.

Κώδικας 3.15 - Αφαίρεση ειδικών χαρακτήρων

```
public static String removeSpecialCharacters(String ocrText){
    Pattern pattern = Pattern.compile("[^a-z0-9 ]",
Pattern.CASE_INSENSITIVE);
    StringBuffer buffer = new StringBuffer(ocrText);
    Matcher matcher = pattern.matcher(buffer);
    while (matcher.find()){
        try {
            buffer.replace(matcher.start(), matcher.end(),"");
        } catch (StringIndexOutOfBoundsException ignored) {}
        matcher = pattern.matcher(buffer);
    }
    return buffer.toString();
}
```

Όταν έχουμε εντοπίσει λανθασμένα μια περιοχή κειμένου, τότε η τεχνική OCR θα επιχειρήσει να εξάγει κάποιο κείμενο, παρ' όλο που μπορεί να μην υπάρχει κανένα. Σε αυτές τις περιπτώσεις συνήθως, τα αποτελέσματα εξαγωγής περιέχουν μια σειρά από τυχαίους ειδικούς χαρακτήρες και όχι κάποιο σύνολο συνηθισμένων απλών χαρακτήρων που είναι και αυτοί που ψάχνουμε κυρίως. Έτσι, με την παρούσα τεχνική, μειώνουμε το κόστος του λανθασμένου εντοπισμού περιοχής κειμένου.

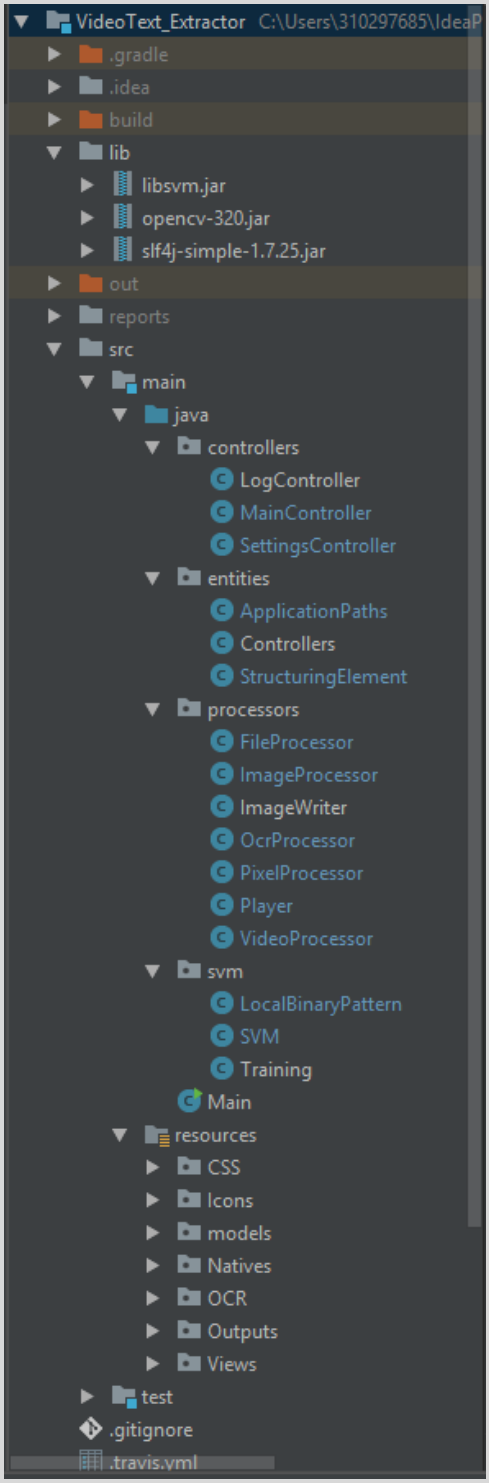
Κώδικας 3.16 - Ορθογραφικός έλεγχος

```
public static String checkForSpelling(String ocrText) {
    List<RuleMatch> matches;
    StringBuffer buffer = new StringBuffer(ocrText);
    int iterations = 0;
    while (true) {
        if (++iterations > 3) return buffer.toString();
        try { matches = languageTool.check(buffer.toString()); }
        catch (IOException e) { return ocrText; }
        if (!matches.isEmpty()){
            if (!matches.get(0).getSuggestedReplacements().isEmpty()){
                try {
                    buffer.replace(
                                matches.get(0).getFromPos(),
                                matches.get(0).getToPos(),
                                matches.get(0).
                                     getSuggestedReplacements().get(0));
                } catch (StringIndexOutOfBoundsException ignored) {}
            } else {
                return " ";
            }
        } else {
            return buffer.toString();
        }
    }
}
```

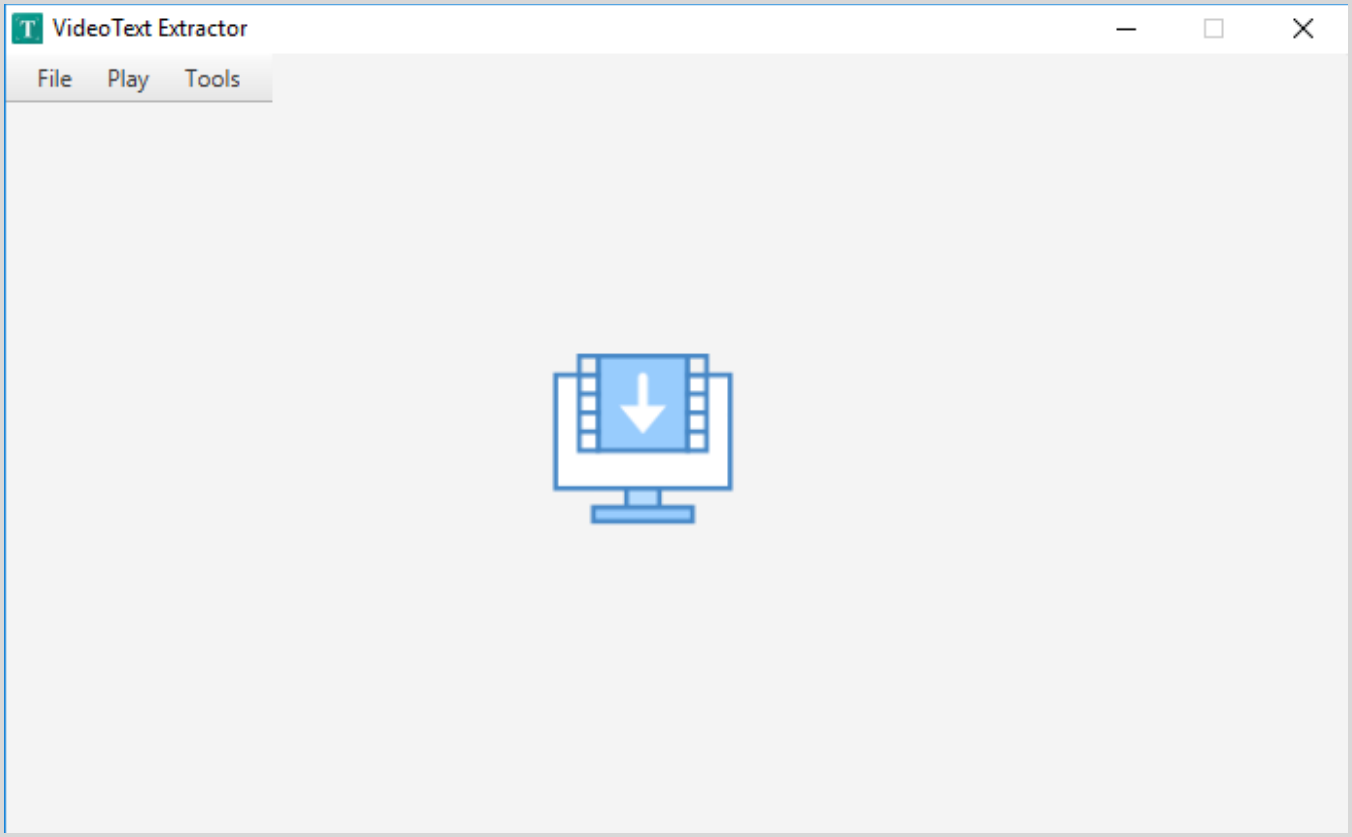
Στον παραπάνω κώδικα χρησιμοποιούμε την βιβλιοθήκη **languagetool**, η οποία μας παρέχει την δυνατότητα να ελέγξουμε άμα οι λέξεις μας είναι ορθογραφικά ορθές και σε περίπτωση που δεν είναι, να τις αντικαταστήσουμε με την κοντινότερη λέξη που προτείνεται από το λεξικό. Στην περίπτωση που έχουμε εντοπίσει μια περιοχή κειμένου που δεν έχει προφτάσει να ολοκληρωθεί ή η αναπαράσταση του κειμένου δεν είναι καθαρή, τότε η παρούσα τεχνική αποδεικνύεται πολύ χρήσιμη.

3.3 ΠΑΡΟΥΣΙΑΣΗ ΕΦΑΡΜΟΓΗΣ

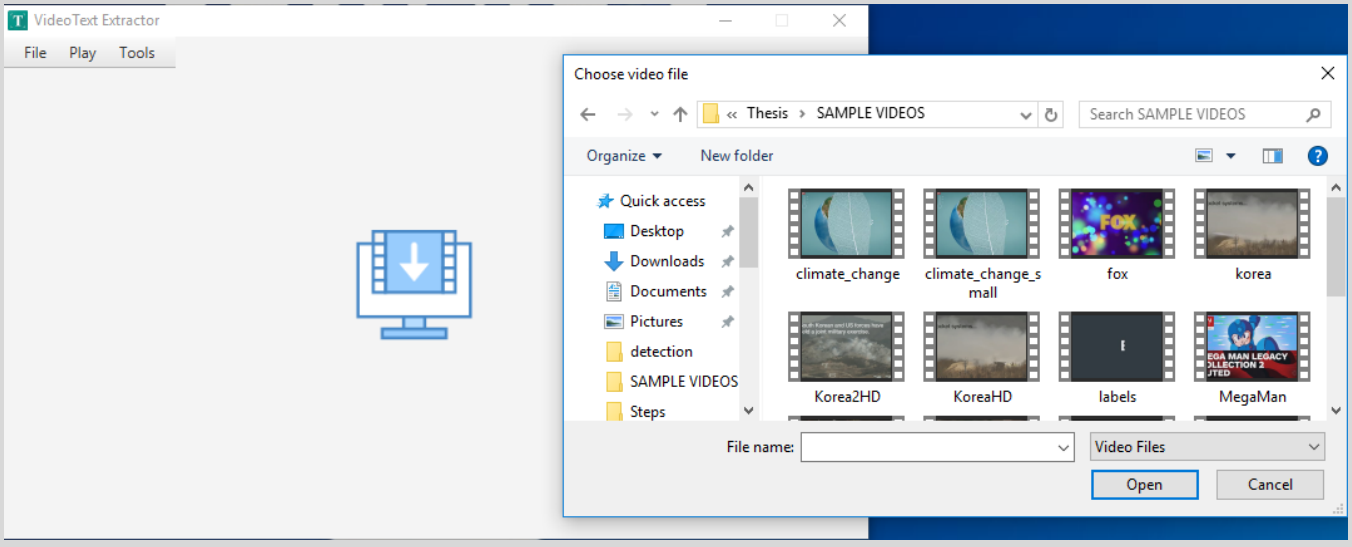
Εικόνα 3.2- Δομή Εφαρμογής



Εικόνα 3.3 - Αρχικό Περιβάλλον

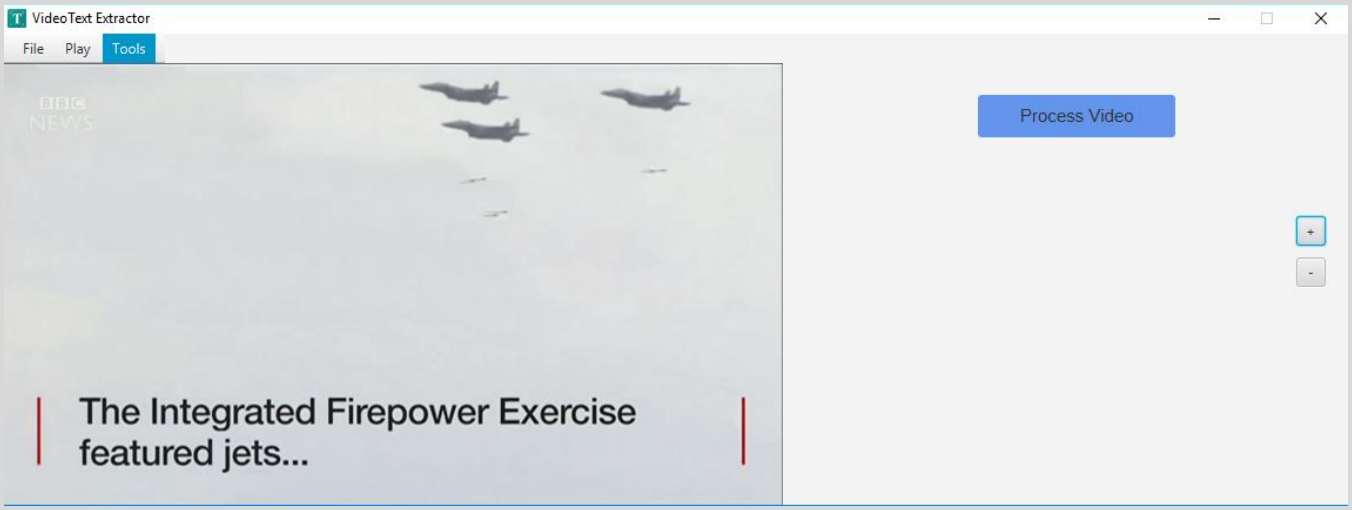


Εικόνα 3.4 - Επιλογή Βίντεο

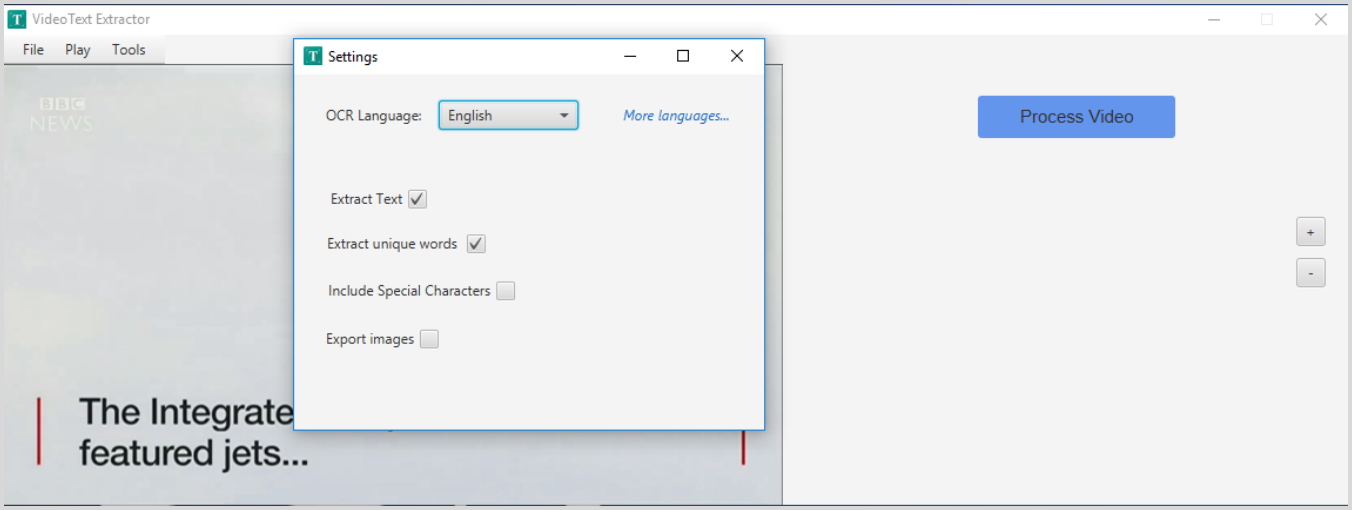


Η επιλογή βίντεο, μπορεί να γίνει μέσω του βοηθητικού παραθύρου του λειτουργικού συστήματος όπως βλέπουμε παραπάνω, ή μέσω drag and drop του αρχείου βίντεο «πάνω» στην εφαρμογή.

Εικόνα 3.5 - Φόρτωση Βίντεο



Εικόνα 3.6 - Μενού Ρυθμίσεων



Το μενού ρυθμίσεων, μας παρέχει διάφορες χρήσιμες επιλογές και λειτουργίες. Η πρώτη, αφορά την επιλογή της φυσικής γλώσσας που επιθυμούμε να εξάγουμε. Η τεχνική OCR βασίζεται στην εκπαίδευση χαρακτήρων και δεδομένου ότι οι χαρακτήρες διαφέρουν αρκετά ανά τα αλφάβητα γλωσσών, είναι λογικό ότι θα πρέπει να δηλώσουμε ποια είναι η γλώσσα εξαγωγής κειμένου ώστε να έχουμε σωστά αποτελέσματα. Η εφαρμογή μας, παρέχει την δυνατότητα επιλογής όλων τα δημοφιλών φυσικών γλωσσών. Ο χρήστης θα πρέπει να επιλέξει την αντίστοιχη γλώσσα ανάλογα με την γλώσσα που αναπαριστά το πολυμεσικό του αρχείο. Η μεθοδολογία μας, αρχικά εντοπίζει τις περιοχές κειμένου και έπειτα επιχειρεί την εξαγωγή του κειμένου που υπάρχει σε αυτές, για αυτόν τον λόγο δίνεται η δυνατότητα, μέσω της δεύτερης επιλογής, να γίνει μόνο ο εντοπισμός του κειμένου (ταχύτερο) εάν αυτό

επιθυμείται από τον χρήστη. Η τρίτη επιλογή αφορά την αποφυγή ή μη ειδικών χαρακτήρων κατά την διαδικασία εξαγωγής.

Τελευταίο και σημαντικότερο, μέσω της τελευταίας επιλογής, πραγματοποιείται η δημιουργία και αποθήκευση αρχείων, τα οποία παρουσιάζουν λεπτομερώς όλα τα βήματα της μεθοδολογίας που εφαρμόστηκε κατά την επεξεργασία του βίντεο.

Αναλυτικότερα, δημιουργούνται και αποθηκεύονται για κάθε βίντεο:

- Όλα τα πλαίσια του βίντεο που επεξεργάστηκαν, με τις εντοπισμένες περιοχές κειμένου ζωγραφισμένες

- Όλα τα βήματα της μεθοδολογίας που εφαρμόστηκε για κάθε πλαίσιο, μία εικόνα για κάθε βήμα

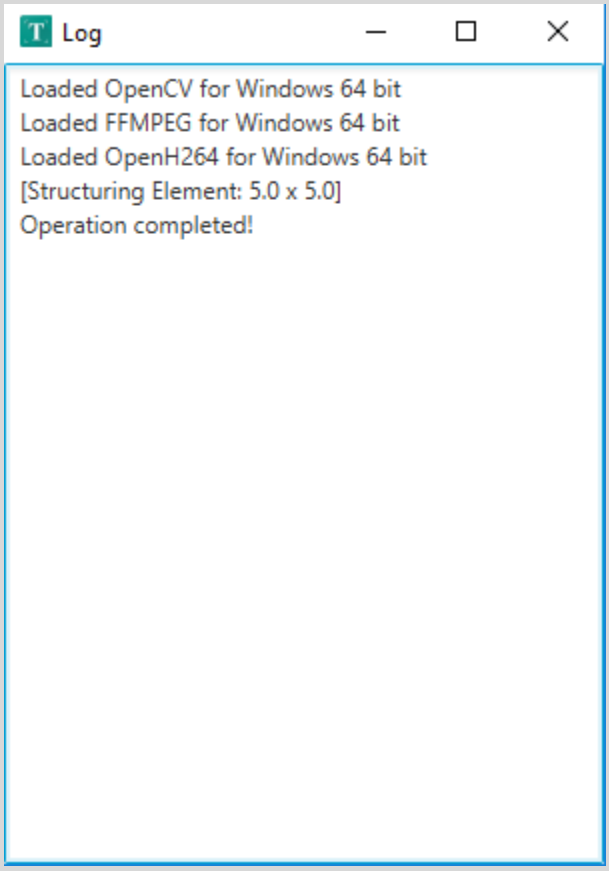
- Τα βήματα της προεπεξεργασίας OCR

- Ένα αντίγραφο του βίντεο, με τις εντοπισμένες περιοχές κειμένου, το οποίο μπορεί να παραχθεί και μέσω της εφαρμογής

- Οι υπο-εικόνες που αποκόπηκαν από το κάθε στάδιο φιλτραρίσματος

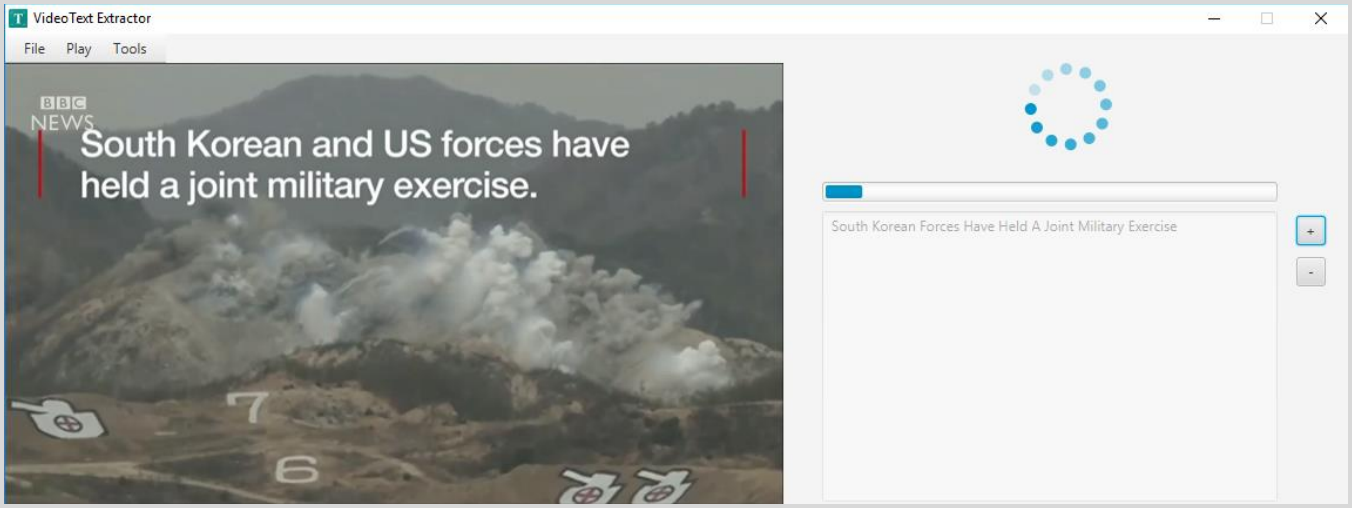
- Τις εικόνες που απέρριψε το μοντέλο SVM και αυτές που αποδέχθηκε

Εικόνα 3.7 - Παράθυρο Ειδοποιήσεων

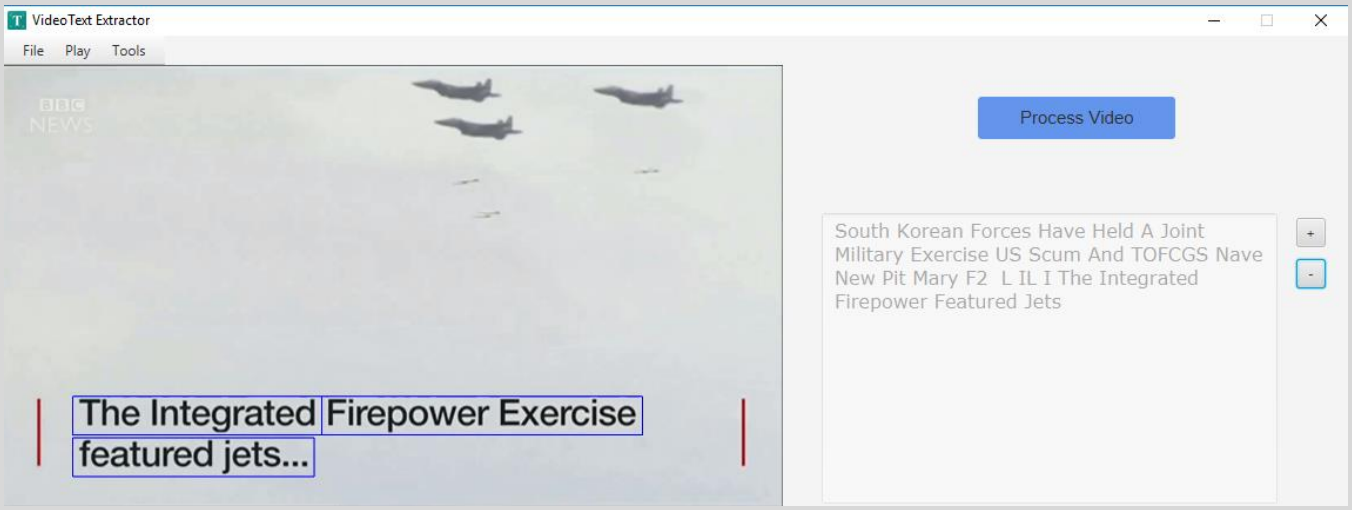


Μέσω του παραθύρου μηνυμάτων μπορούμε να έχουμε άμεση εικόνα και για την κατάσταση της εφαρμογής, να ενημερωθούμε για πιθανά σφάλματα του λογισμικού που ίσως υπήρξαν, χρήσιμα μηνύματα όπως εάν φορτώθηκαν επιτυχώς ή απαραίτητες βιβλιοθήκες ή και ποιο δομικό στοιχείο επιλέχθηκε για την μορφολογική διαστολή, πράγμα που εξαρτάται από τις διαστάσεις του βίντεο.

Εικόνα 3.8 - Επεξεργασία βίντεο



Εικόνα 3.9 - Ολοκλήρωση επεξεργασίας

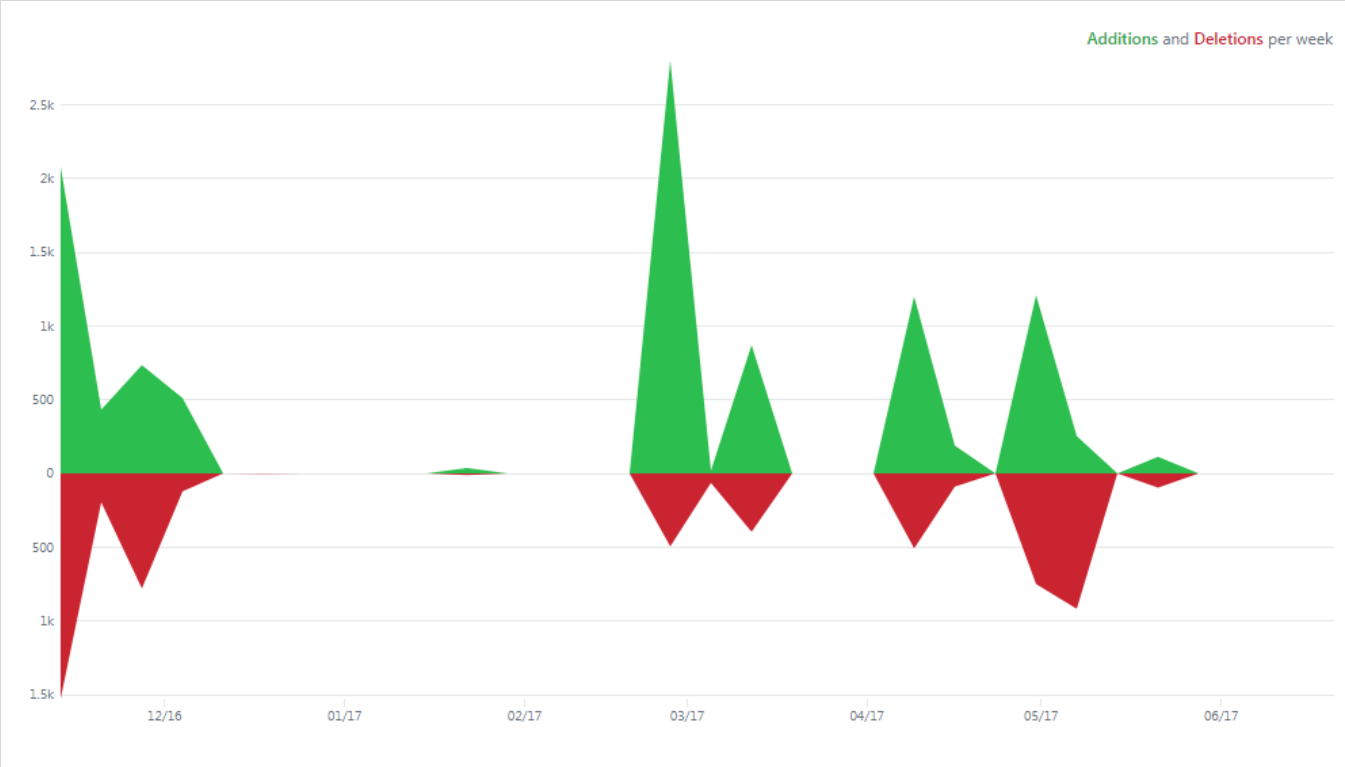


Δεξιά φαίνονται τα αποτελέσματα εξαγωγής του βίντεο. Μπορούμε να αυξομειώσουμε το μέγεθος της γραμματοσειράς, χρησιμοποιώντας τα δύο κουμπιά που βρίσκονται δεξιότερα. Το κείμενο εξαγωγής, είναι επεξεργάσιμο, το οποίο σημαίνει ότι μπορεί να γίνει αντιγραφή - επικόλληση για οποιαδήποτε περαιτέρω επεξεργασία.

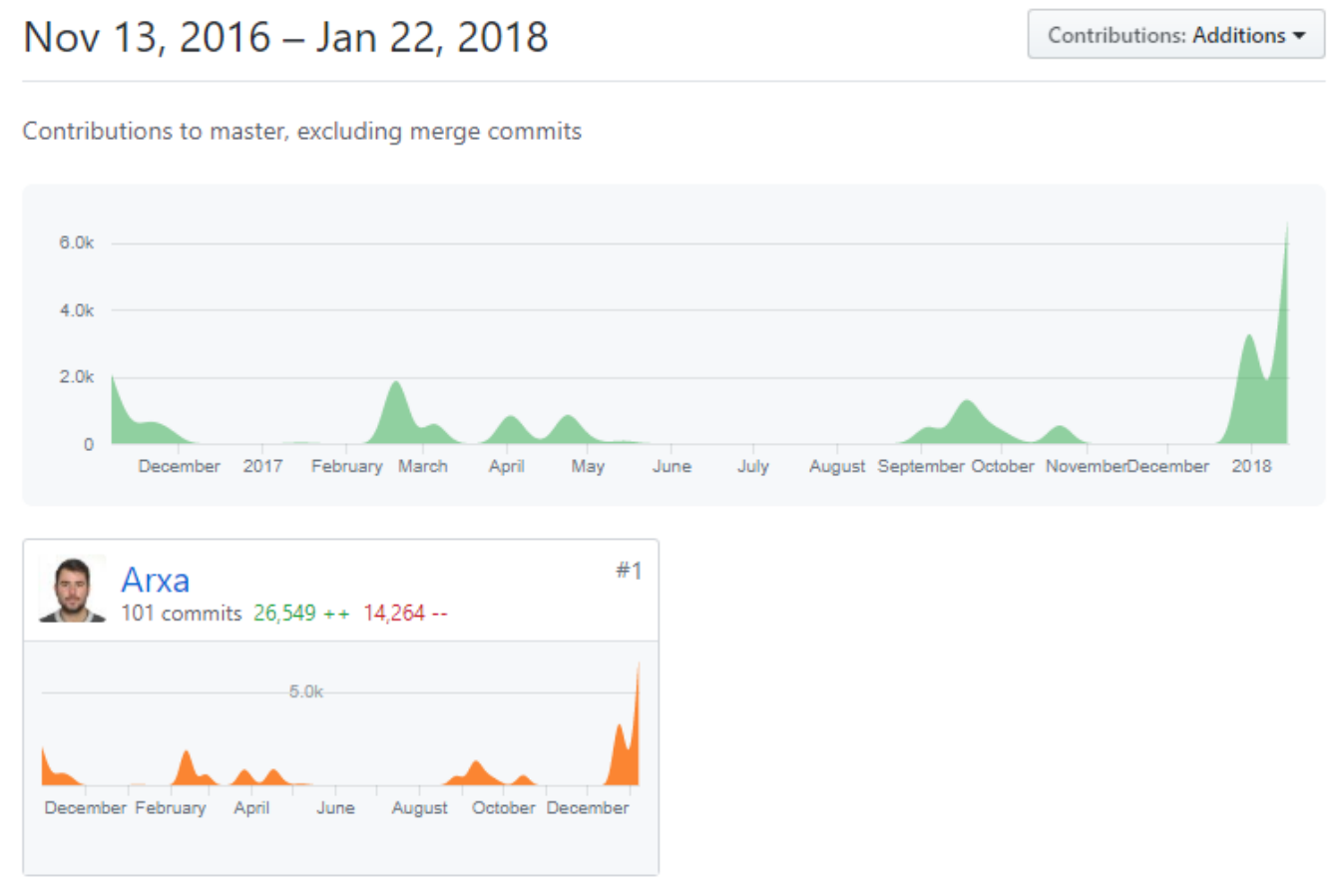
3.4 ΓΡΑΦΗΜΑΤΑ

Όπως αναφέραμε και στην Ενότητα 3.1, έχουμε χρησιμοποιήσει την τεχνολογία Git για το Version Control της εφαρμογής. Μέσω της ιστοσελίδας GitHub, έχουμε παράξει τα παρακάτω χρήσιμα γραφήματα:

Εικόνα 3.10 - Συχνότητα Προσθαφαίρεσης Κώδικα



Εικόνα 3.11 - Συνεισφορά Κώδικα



Η ανάθεση της πτυχιακής πραγματοποιήθηκε τον Οκτώβρη του 2016. Όντως, σύμφωνα με το παραπάνω διάγραμμα, η προσθαφαίρεση κώδικα ξεκινάει περίπου εκείνη την ημερομηνία. Από αυτό το παραπάνω διάγραμμα μπορούμε να συμπεράνουμε πως η «μεικτή» διάρκεια της Πτυχιακής υπολογίζεται περίπου στους 14 μήνες, ενώ η «καθαρή» διάρκεια περίπου στους 7 μήνες. Συνολικά προστέθηκαν 26.549 γραμμές κώδικα ενώ αφαιρέθηκαν 14.264, εκ των οποίων εξάγουμε τον αριθμό 12.285 που είναι οι τελικές γραμμές κώδικα της εφαρμογής.

4 ΕΠΙΛΟΓΟΣ

4.1 ΛΙΓΑ ΛΟΓΙΑ

Οι τεχνολογικές απαιτήσεις και προσδοκίες μας ως κοινωνία εξελίσσονται ραγδαία. Μπορούμε να δούμε ξεκάθαρα τον πρωταγωνιστικό και κρίσιμο ρόλο που κατέχουν η Ψηφιακή Επεξεργασία Εικόνας και η Μηχανική Μάθηση στις ζωές μας σήμερα, αλλά και πόσο εντονότερη θα γίνει η παρουσία των τεχνολογιών αυτών τα επόμενα χρόνια. Για παράδειγμα, έχει ήδη αναπτυχθεί λογισμικό, το οποίο εξοπλίζει ένα απλό smartphone κινητό τηλέφωνο με την ικανότητα να αξιολογεί μέσω της κάμεράς του, περιοχές του δέρματος για την ύπαρξη ή όχι μελανώματος [26]. Επίσης, το 2017, η εταιρία κατασκευής αυτόνομων οχημάτων *Waymo,* ανακοίνωσε ότι το αυτόνομο όχημά της, έχει καταφέρει να οδηγήσει 560.000 χιλιόμετρα, σημειώνοντας μονάχα 63 εμπλοκές (*στατιστικό μέτρο σύγκρισης κατασκευαστών αυτόνομων οχημάτων*) [29]. Παρόμοια παραδείγματα, που αντιπροσωπεύουν την εξέλιξη και την σημαντικότητα της ΨΕΕ και της Μηχανικής Μάθησης, υπάρχουν πολλά.

Ο εντοπισμός κειμένου αποτελεί βασικό χαρακτηριστικό των τεχνολογιών που προαναφέραμε, επομένως η εξέλιξη και η μελέτη της ιδιότητας αυτής είναι πολύ σημαντική. Για αυτό τον λόγο, το περιεχόμενο της Πτυχιακής που παρουσιάστηκε, είναι επίκαιρο και ουσιαστικό, καθώς εξετάζει σε βάθος αυτή την σημαντική τεχνολογική ικανότητα.

Το λογισμικό που αναπτύχθηκε, προσφέρει εκπαιδευτική πληροφορία στον χρήστη, όσον αφορά την μεθοδολογία που υλοποιήθηκε, την ανάλυση της εικόνας και τον εντοπισμό του κειμένου αυτής. Πέρα από την εκπαιδευτική χρήση, η εφαρμογή μπορεί επίσης εύκολα να τροποποιηθεί για επαγγελματική ή και επιστημονική χρήση. Ο εντοπισμός και η εξαγωγή κειμένου που αναπαρίσταται σε βίντεο, είναι ένα εξαιρετικό και πολύ χρήσιμο εργαλείο, που μπορεί να εφαρμοστεί σε διάφορες περιπτώσεις. Ένα ενδιαφέρον παράδειγμα εφαρμογής, είναι η αναζήτηση λέξεων-κλειδιών, που αναγράφονται στον πίνακα του καθηγητή, σε διαδικτυακά βιντεοσκοπημένα μαθήματα. Με το ίδιο σκεπτικό, μπορούμε να φανταστούμε διάφορες άλλες χρήσεις της προσέγγισης αυτής, όπου ο γενικός στόχος θα είναι η εξόρυξη γνώσης και πληροφορίας από ψηφιακά πολυμεσικά αρχεία που περιέχουν κείμενο.

4.2 ΑΝΑΦΟΡΕΣ

[1] R. Gonzalez and R. Woods Digital Image Processing, Addison-Wesley Publishing Company, 1992, Chap. 7.

[2] M. Sezgin & B. Sankur (2004). "Survey over image thresholding techniques and quantitative performance evaluation". Journal of Electronic Imaging. 13 (1): 146–165.

[3] Shapiro, Linda G. & Stockman, George C. (2002). "Computer Vision". Prentice Hall. ISBN 0-13-030796-3

[4] Shapiro, L. G. & Stockman, G. C: "Computer Vision", page 137, 150. Prentice Hall, 2001

[5] MacKay, David (2003). "Chapter 20. An Example Inference Task: Clustering" (PDF). Information Theory, Inference and Learning Algorithms. Cambridge University Press. pp. 284–292

[6] Image Analysis and Mathematical Morphology by Jean Serra, ISBN 0-12-637240-3 (1982)

[7] R. Gonzales and R. Woods Digital Image Processing, Addison-Wesley Publishing Company, 1992, pp 443 - 452.

[8] Borden Dent (Cartography--Thematic Map Design, Fifth Edition, 1999, pages 147-149)

[9] T. Ojala, M. Pietikäinen, and D. Harwood (1996), "A Comparative Study of Texture Measures with Classification Based on Feature Distributions", Pattern Recognition, vol. 29, pp. 51-59.

[10] Chandola, V.; Banerjee, A.; Kumar, V. (2009). "Anomaly detection: A survey". ACM Computing Surveys. 41 (3): 1–58

[11] Juszczak, P.; D. M. J. Tax; R. P. W. Dui (2002). "Feature scaling in support vector data descriptions". Proc. 8th Annu. Conf. Adv. School Comput. Imaging: 25–30.

[12] Cortes, Corinna; Vapnik, Vladimir N. (1995). "Support-vector networks". Machine Learning. 20 (3): 273–297

[13] Palaiahnakote Shivakumara, Trung Quy Phan and Chew Lim Tan, Senior Member, IEEE "A Laplacian Approach to Multi-Oriented Text Detection in Video", IEEE

[14] Trung Quy Phan, Palaiahnakote Shivakumara and Chew Lim Tan "A Laplacian Method for Video Text Detection", School of Computing, National University of Singapore, 2009

[15] Rakesh Mehta, Karen Egiazarian, "Rotated Local Binary Pattern (RLBP) Rotation invariant texture descriptor", 2nd International Conference on Pattern Recognition Applications and Methods, ICPRAM 2013, Barcelona, Spain, 2013

[16] Cong Yao, "MSRA Text Detection 500 Database (MSRA-TD500)", Huazhong University of Science and Technology, 2012

[17] Visual Geometry Group, "Synthetic Word Dataset ", Department of Engineering Science, University of Oxford

[18] https://stackoverflow.com/questions/9480013/image-processing-to-improve-tesseract-ocr-accuracy

[19] https://en.wikipedia.org/wiki/Optical_character_recognition

[20] https://stackoverflow.com/questions/10792576/libsvm-java-implementation

[21] https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

[22] https://www.quora.com/Does-the-parameter-C-affect-one-class-SVM

[23] https://ninghang.blogspot.co.uk/2012/11/list-of-mat-type-in-opencv.html

[24] https://stackoverflow.com/questions/29108270/opencv-2-4-10-bwlabel-connected- components

[25] https://docs.gimp.org/en/plug-in-unsharp-mask.html

[26] https://itunes.apple.com/gb/app/skinvision-detect-skin-cancer/id545293136?mt=8

[27] https://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/

[28] https://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm

[29] "Uber' self-driving system was still 400 times worse Waymo in 2018 on key distance intervention metric | NextBigFuture.com". NextBigFuture.com. 2018-03-25. Retrieved 2018-03-25.

[30] https://stackoverflow.com/a/20679213/5165833