Άσκηση 3η – Μηχανική Όραση

Σοφία Καφρίτσα Γεωργαντά, 2016030136

- *Σκοπός*

Σε αυτή την άσκηση υλοποίησα ένα object detector βασισμένο σε gradient features and sliding window classification. Παρακάτω, παραθέτω τις αλλαγές και τις προσθήκες που έκανα στις δεδομένες συναρτήσεις.

- *Επεξεργασία*

1. Image Gradient: συνάρτηση mygradient()

Χρησιμοποιώντας ως derivative filter το F, για τον άξονα x και το αντίστροφο του F για τον άξονα y, δημιούργησα τις μεταβλητές dx, dy, για να βρω το μέγεθος και τον προσανατολισμό των εισαγόμενων εικόνων. Χρησιμοποίησα τη συνάρτηση imfilter με παράμετρο replicate και υπολόγισα τα στοιχεία ως εξής:

```
mag = sqrt(dx.^2 + dy.^2);
ori = atan2(dy, dx).*-180/pi;
```

Στη συνέχεια οπτικοποίησα αυτά τα δεδομένα χρησιμοποιώντας τη συνάρτηση *imagesc*.

2. Histograms of Gradient Orientations: συνάρτηση hog()

Η συνάρτηση αυτή παίρνει ως όρισμα την εικόνα με διαστάσεις h,w και υπολογίζει τα gradient orientation ιστογράμματα, πάνω σε 8x8 block από pixels. Αυτό το εξασφαλίζουμε από το τελικό μέγεθος ορίζοντας `h2 = ceil(h/8),w2 = ceil(w/8)`.
Επίσης, όπως ζητάει η εκφώνηση ορίζω 9 orientation bins. Στη συνέχεια, καλώ τη συνάρτηση mygradient(), μέσω της οποίας βρίσκω τα magnitude και orientation. Χρησιμοποιώ το magnitude για να ορίσω το threshold. Η διαδικασία που εφάρμοσα γίνεται με for loop πάνω σε κάθε orientation bin. Μέσα σε αυτό το loop, υπάρχει ένα δεύτερο, το οποίο διατρέχει την εικόνα και εντοπίζει τα pixels εκείνα που βρίσκονται πάνω από το threshold που έχω ορίσει και ταυτόχρονα ο προσανατολισμός τους βρίσκεται στο δεδομένο bin. Η διαδικασία φαίνεται παρακάτω και βλέπουμε ότι το τμήμα από -p/2 έως p/2 είναι χωρισμένο σε p/9. Κάθε φορά που βρίσκω κάποιο pixels ακμής, εισάγω στην binary εικόνα B το '1'.

```
for x = 1:h
    for y = 1:w
        if(((-pi/2)+(i-1)*pi/9) < ori(x,y) && ori(x,y) <= ((-pi/2)+(pi/9*i)) && mag(x,y) > thresh)
            B(x,y) = 1;
        end
    end
end
```

Στο τέλος του αρχικού for loop, αποθηκεύω τις τιμές της B σε 8x8 pixel blocks και δημιουργώ το ιστόγραμμα. Χρησιμοποιώ τις συναρτήσεις *im2col* και *reshape.*
Μετά, κανονικοποιώ το ιστόγραμμα, ώστε το άθροισμα πάνω σε κάθε orientation bin να είναι 1 για κάθε block και ελέγχω να μην διαιρέσω με το μηδέν. Η διαδικασία φαίνεται παρακάτω:

```
total = zeros(h2,w2);
for i = 1:nori
    for x = 1:h2
        for y = 1:w2
           total(x,y) = total(x,y) + ohist(x,y,i);
        end
    end
end

for i = 1:nori
    for x = 1:h2
        for y = 1:w2
            ohist(x,y,i) = ohist(x,y,i) ./ total(x,y);
        end
    end
end
ohist(isnan(ohist)) = 0;
```

3. Detection: συνάρτηση detect()

Αυτή η συνάρτηση γυρίζει τον αριθμό που έχουμε ορίσει από top detections, εφαρμόζοντας στην εικόνα ένα template. Αρχικά, υπολογίζω το feature map της εικόνας με τη συνάρτηση hog που δημιούργησα προηγουμένως. Εφαρμόζω cross-correlation όπως φαίνεται στην παρακάτω εικόνα, φιλτράροντας κάθε orientation μεμονωμένα, και τέλος προσθέτοντάς τα. Το τελικό αποτέλεσμα έχει μέγεθος (h/8)x(w/8)x9.

```
R = zeros(size(f,1),size(f,2)); %(h/8)X(w/8)
for i = 1:nori
  R = R + imfilter(f(:,:,i), template, 'replicate');
end
```

Έπειτα, εφαρμόζω τη διαδικασία που αναλύεται στην εκφώνηση για non-maxima suppression, ταξινομώντας τα detections και ελέγχοντας αν η τοποθεσία κάθε detection που βάζω στη λίστα με τα detections βρίσκεται αρκετά κοντά με κάποιο άλλο detection που ήδη βρίσκεται στη λίστα. Συγκεκριμένα, αν το νέο detection είναι πιο κοντά από απόσταση λιγότερη από το 70% των διαστάσεων του template, τότε υπάρχει επικάλυψη. Αυτό το ελέγχω με τη Boolean μεταβλητή overlap, η οποία είναι true αν εντοπίζεται αυτή η επικάλυψη ή false αν δεν εντοπίζεται, όπου τότε το καταχωρώ στην τελική λίστα. Παρακάτω φαίνεται η ανάθεση τιμής στην overlap.

```
overlap = any(sqrt((x-xpixel).^2 + (y-ypixel).^2) < 0.7 * 8 * size(template,1));
```

Στην παρακάτω εικόνα, προσθέτω το detection στη λίστα με τα υπόλοιπα, σε περίπτωση που δεν παρατηρήθηκε επικάλυψη.

```
    if (~overlap)
      detcount = detcount+1;
      x(detcount) = xpixel;
      y(detcount) = ypixel;
      score(detcount) = 0;
    end
    i = i + 1
  end
```

*Για να βρω τις συντεταγμένες του block χρησιμοποιώ τη συνάρτηση *ind2sub* και έπειτα μετατρέπω τις συντεταγμένες των block σε συντεταγμένες pixels πολλαπλασιάζοντας επί 8.

4. Detection Script : detect_script():

Σε αυτό το demo script έκανα κάποιες αλλαγές, ώστε να αντικαταστήσω τα hard-coded κομμάτια και να προσθέσω τα negative training examples. Αρχικά, ορίζω αριθμό από positive και negative clicks και στη συνέχεια ορίζω template size ίσο με 16, καθώς έπειτα από δοκιμές θεώρησα ότι είναι το πιο κατάλληλο. Επίσης, ορίζω τη μεταβλητή block που ισούται με 8 και στη συνέχεια υπολογίζω το 8x8 block το οποίο πάτησε ο χρήστης, με τις μεταβλητές blockx, blocky. Στη συνέχεια, όπως φαίνεται παρακάτω, οπτικοποιώ το τετράγωνο γύρω από την τοποθεσία που πάτησε ο χρήστης. Σε περίπτωση που έχω template size ίσο με 16, το τετράγωνο αυτό θα περιέχει 128 pixels:

```
for i = 1:nclick
  patch = Itrain(block*blocky(i)+(-(block^2-1):block^2),block*blockx(i)+(-(block^2-1):block^2));
  figure(3); subplot(3,2,i); imshow(patch);
end
```

Έπειτα, καλώ τη συνάρτηση hog για να υπολογίσω τα hog features και με βάση αυτά, υπολογίζω το template για τα user clicks:

```
  postemplate = zeros(template_size,template_size,9);
for i = 1:nclick
   postemplate = postemplate + f(blocky(i)+(-(block-1):block),blockx(i)+(-(block-1):block),:);
end
  postemplate = postemplate/nclick;
```

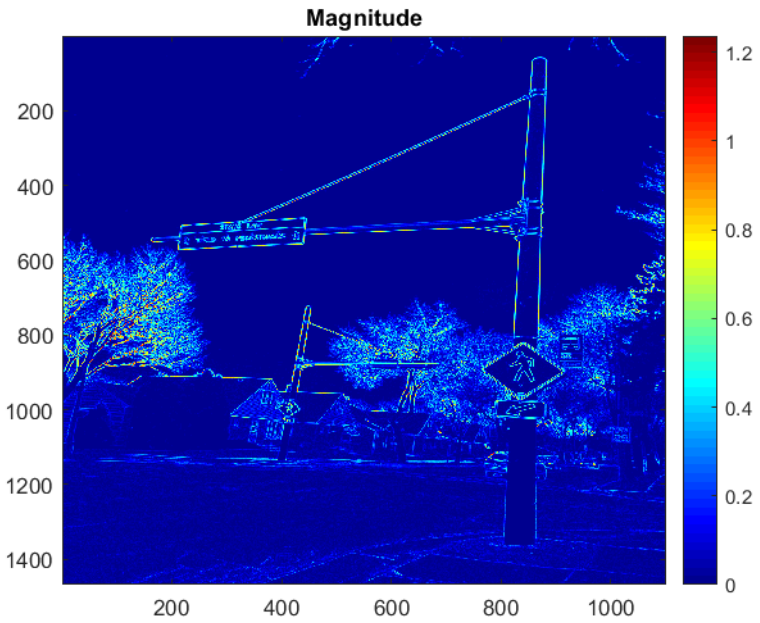
Αντίστοιχη διαδικασία εφαρμόζω και για τα negative clicks. Τέλος, για να παράξω το τελικό template, αφαιρώ από τα θετικά average templates τα αρνητικά. Στη συνέχεια, εισάγω μία εικόνα testing, και ορίζω τον αριθμό των detections που θέλω στο τελικό αποτέλεσμα. Για τα αποτελέσματα έχω κρατήσει την default τιμή που είναι το 8. Τέλος, χρησιμοποιώντας τη συνάρτηση rectangle, εμφανίζω τα detections, με διαφορετικά χρώματα ανάλογα με το score.

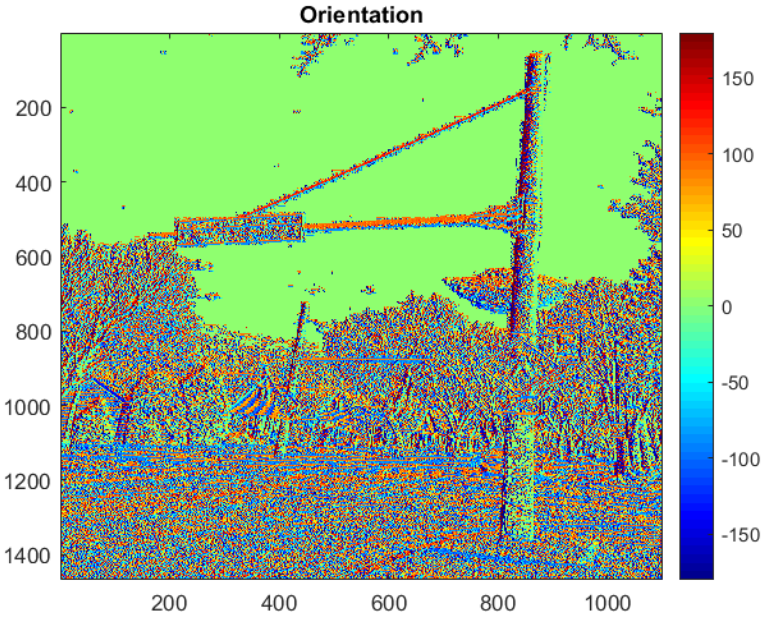
```
% find top 8 detections in Itest
ndet = 8;
[x,y,score] = detect(Itest,template,ndet);
ndet = length(x);

%display top ndet detections
figure(9); clf; imshow(Itest);
for i = 1:ndet
  % draw a rectangle.  use color to encode confidence of detection
  %  top scoring are green, fading to red
  hold on;
  h = rectangle('Position',[x(i)-(block^2) y(i)-(block^2) (template_size*block) (template_size*block)],'EdgeColor',[(i/ndet) ((ndet-i)/ndet)  0],'LineWidth',3,'Curvature
  hold off;
end
```

- *Ανάλυση Αποτελεσμάτων*

Για το πρώτο παράδειγμα με την πινακίδα, από την παρακάτω εικόνα εξάγω τα positive και negative templates που φαίνονται παρακάτω:

Παρακάτω, βλέπουμε τις οπτικοποιήσεις των Gradient Magnitude και Gradient Orientation της testing εικόνας.

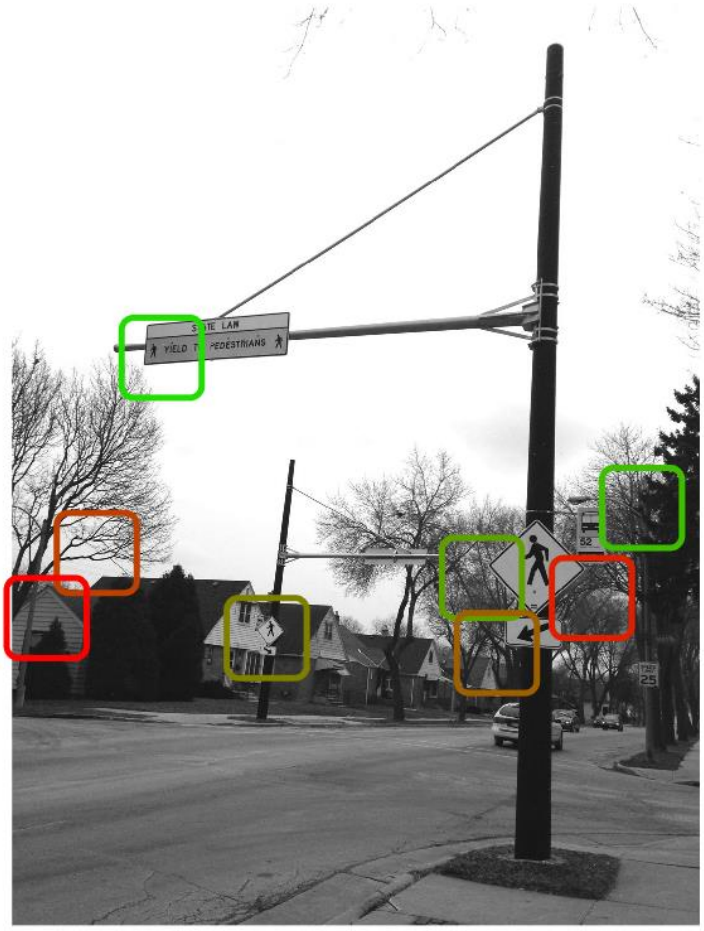




Στη συνέχεια, φαίνονται οι οπτικοποιήσεις των HOG της αρχικής εικόνας και της εικόνας που χρησιμοποιώ για testing.

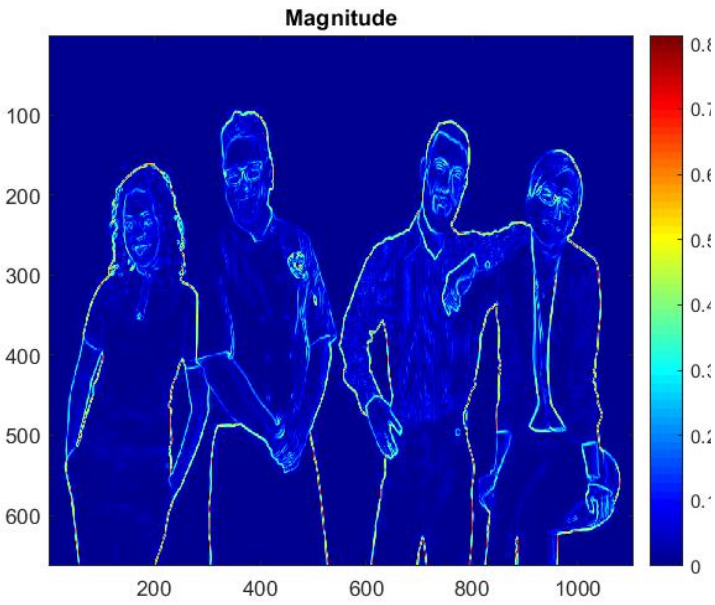
Τέλος, δίπλα παραθέτω το τελικό αποτέλεσμα με τα detections.

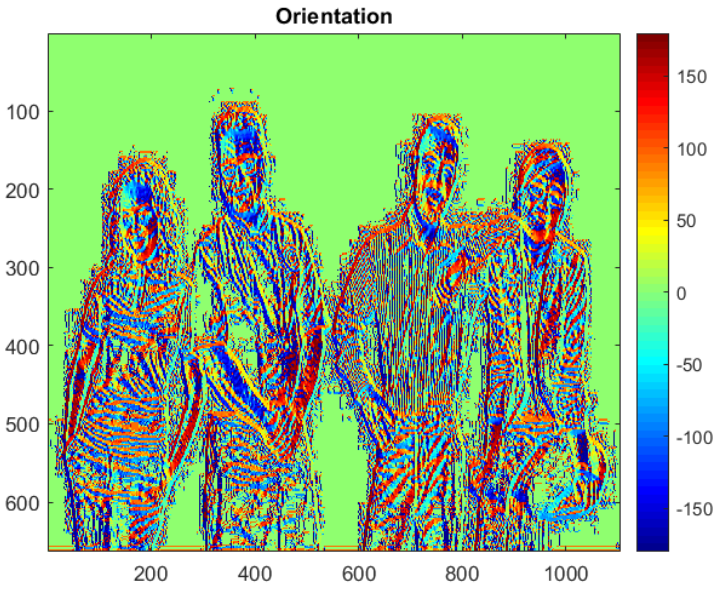
Αυτό που παρατηρούμε είναι ότι δεν εντοπίζονται όλα τα αντικείμενα, και κυρίως αυτά που έχουν μεγαλύτερη διαφορά στην οπτική γωνία από το αρχικό αντικείμενο. Επίσης, εντοπίζονται και κάποια σημεία που ανήκουν σε πινακίδες, αλλά δεν αποτυπώνουν τον «άνθρωπο».



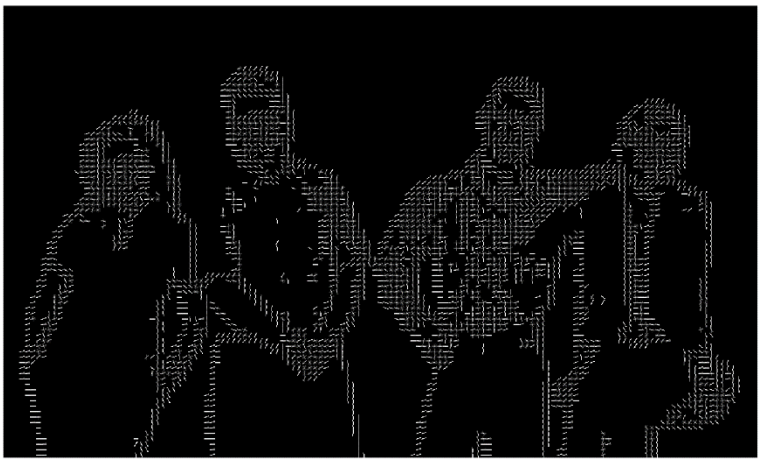
Για το δεύτερο set από εικόνες, αύξησα τα clicks καθώς έχουμε πολλά πρόσωπα. Παρακάτω φαίνεται η αρχική εικόνα, τα θετικά και αρνητικά templates.

Παρακάτω, βλέπουμε τις οπτικοποιήσεις των Gradient Magnitude και Gradient Orientation της testing εικόνας.





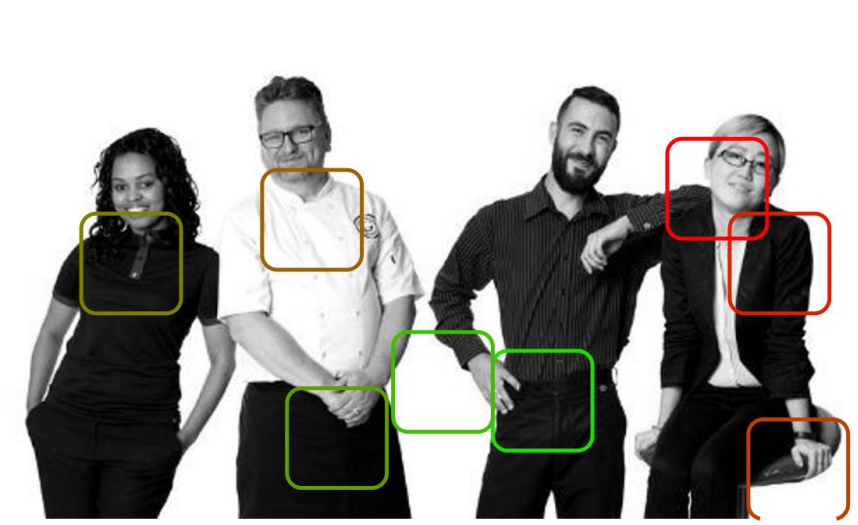
Στη συνέχεια, φαίνονται οι οπτικοποιήσεις των HOG της αρχικής εικόνας και της εικόνας που χρησιμοποιώ για testing.



Τέλος, βλέπουμε το αποτέλεσμα με τα detections. Τα αποτελέσματα είναι ικανοποιητικά.

Σε μία άλλη δοκιμή, επέλεξα για positive templates πρόσωπα που βρίσκονται ανάποδα, δηλαδή κάποια πρόσωπα που βρίσκονται στο κάτω μέρος του κύκλου που έχουν σχηματίσει οι άνθρωποι στην εικόνα, όπως φαίνεται δίπλα.

Το αποτέλεσμα ήταν να μην εντοπιστούν τόσο αποτελεσματικά τα πρόσωπα στην testing εικόνα, επομένως ο αλγόριθμος είναι σχετικά ευαίσθητος ως προς τον προσανατολισμό.



- Ως γενικές παρατηρήσεις, φαίνεται ότι τα αντικείμενα που πρέπει να εντοπιστούν χρειάζεται να έχουν παρόμοιο magnitude και orientations με το αντικείμενο αναφοράς, προκειμένου να λειτουργήσει αποτελεσματικά ο αλγόριθμος. Ακόμη, από δοκιμές φαίνεται ότι το μέγεθος του detection patch πρέπει να είναι αρκετά μεγάλο, ώστε να χωράει όση περισσότερη πληροφορία από το αντικείμενο γίνεται, αλλά να μην είναι τόσο μεγάλο ώστε να περιέχει και επιπλέον πληροφορία-θόρυβο, η οποία μπορεί να αποπροσανατολίσει το αποτέλεσμα.
- Χρησιμοποίησα υλικό από τις παρακάτω πηγές:
- https://learnopencv.com/histogram-of-oriented-gradients/
- http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf
- https://medium.com/the-downling/histogram-of-oriented-gradients-hog-heading-classification-a92d1cf5b3cc

- https://github.com/Anbang-Hu/cv/tree/b7882fb28ccc61ebba1427ef50342f0fbecc12ce/5.image_understanding
- https://github.com/abhishekbhatia1/CodeSamples/tree/d972a10e1e759a5d55b76613677b3ad2267976bc/Matlab/ComputerVision/hw5
- https://github.com/zhaotiny/16720-Computer-Vision-Homeworks/tree/33c964a66bf51be180d14b281992d636cf0a0fea/Homework%205/My%20Code
- https://github.com/Skywonder/CS116/tree/1be0cdbb60748218dc439da3d687a09a0439ff72/Detection