VISIÓ PER COMPUTADOR
SHORT PROJECT

Detecció de la mirada

Albert Figuera Pérez
Jacobo Moral Buendía
Quadrimestre tardor 2018-2019

Contents

Objectiu

L'objectiu del projecte és implementar un sistema automàtic per detectar la mirada mitjançant visió per computador. El sistema ha de ser capaç de detectar on són els ulls i posteriorment si aquest estan mirant a la càmera o no. Les imatges seran principalment d'interior i extretes de càmeres per a vídeo-conferències amb l'enquadre usual en aquest tipus d'imatges. Les imatges hauran estat preses amb persones en una orientació vertical amb lleugeres inclinacions del cap.

El conjunt d'imatges de prova han estat extretes de la web de BioID. Aquest conjunt està compost per 1520 imatges en nivells de gris de 384 × 286 pixels i les trobareu al disc de l'assignatura en una carpeta anomenada Short Project. Per a cada imatge trobareu un fitxer text amb un nom similar a BioID_0000.eye amb la informació de la posició dels ulls de la imatge corresponent.

LX	LY	RX	RY
232	110	161	110

En base al total del conjunt d'imatges podeu construir-vos el conjunt d'imatges que serviran per la fase d'aprenentatge, avaluant la bondat del vostre predictor amb la tècnica del validació creuada (k-fold cross validation). Les imatges utilitzades per realitzar el test no haurien d'estar presents en la fase d'aprenentatge. En la fase d'aprenentatge us serà necessari comptar amb unes imatges d'ulls (5%) i unes altres de no ulls (95%). Per obtenir les imatges d'ulls podeu fer servir els índex del fitxer BioID_xxxx.eye i per els no-ulls feu servir imatges aleatòries extretes de la part de les imatge que no són ulls.

Introducció

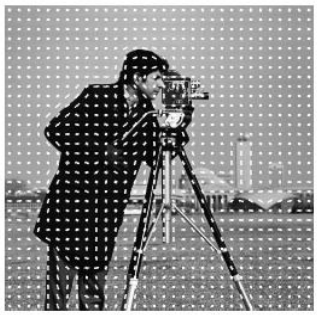
El projecte consisteix en implementar un sistema automàtic per detectar la mirada mitjançant visió per computador per tal de conseguir això utilitzarem diferents algoritmes i eines ja desenvolupades.

Primer de tot haurem d'aconseguir un dataset de cares per tal de dividir-lo en diferents dataset: ulls, no ulls, mirades, no mirades.

Després d'haver separat els datasets i tenir-los ben organitzats haurem d' aconseguir un predictor que funcioni d'una manera considerable, per tal de que funcioni prèviament haurem d'escollir una serie de característiques. Aquestes característiques han d'ésser concretes per aconseguir un bon funcionament del predictor. Això ho haurem de fer primer per detectar ulls i després per detectar si aquests ulls estan mirant a la càmara o no.

Histogram of oriented gradients

L'histograma de gradients orientats (HOG) és un descriptor de característiques utilitzat en la visió per computador i el processament d'imatges amb el propòsit de detectar objectes. La tècnica té en compte les aparicions de l'orientació del gradient en porcions localitzades d'una imatge. quest mètode és similar a altres descriptors visuals, com ara Edge Orientation Histograms (EOH), Scale-invariant feature transform (SIFT) o Shape context (SC), però es diferencia en el fet que es calcula sobre una densa quadrícula de cel·les uniformement espaiades i s'utilitza la superposició de normalització del contrast local per millorar la precisió.



Local binary patterns

Els patrons binaris locals (LBP) són un tipus de descriptor visual utilitzat per a la classificació en la visió per computadora. LBP és el cas particular del model de Texture Spectrum proposat el 1990. LBP va ser descrit per primera vegada el 1994. Des d'aleshores s'ha trobat com una característica potent per a la classificació de la textura; també s'ha determinat que quan el LBP es combina amb el descriptor de degradats orientats per histograma (HOG), millora considerablement el rendiment de la detecció en alguns conjunts de dades.

Random forest

Random forest és un mètode d'aprenentatge conjunt per a la classificació, la regressió i altres tasques que opera mitjançant la construcció d'una multitud d'arbres de decisió en el temps d'entrenament i la distribució de la classe que és el mode de classes (classificació) o predicció mitjana (regressió) dels arbres individuals.

Tree Bagger

TreeBagger empaqueta un conjunt d'arbres de decisió per classificació o regressió. L'estoc d'embalatge representa l'agregació d'arrencada. Cada arbre del conjunt es cultiva en una rèplica d'arrencada independent de la informació d'entrada. Les observacions no incloses en aquesta rèplica són "fora de borsa" per a aquest arbre.

TreeBagger es basa en la funcionalitat ClassificationTree i RegressionTree per al creixement d'arbres individuals. En particular, ClassificationTree i RegressionTree accepta el nombre de funcions seleccionades a l'atzar per cada divisió de decisió com a argument d'entrada opcional. És a dir, TreeBagger implementa l'algorisme random forest

Confusion matrix

Una matriu de confusió és una taula que s'utilitza sovint per descriure el rendiment d'un model de classificació (o "classificador") en un conjunt de dades de prova per als quals es coneixen els valors veritables. La matriu de confusió és relativament senzilla d'entendre, però la terminologia relacionada pot ser confusa.

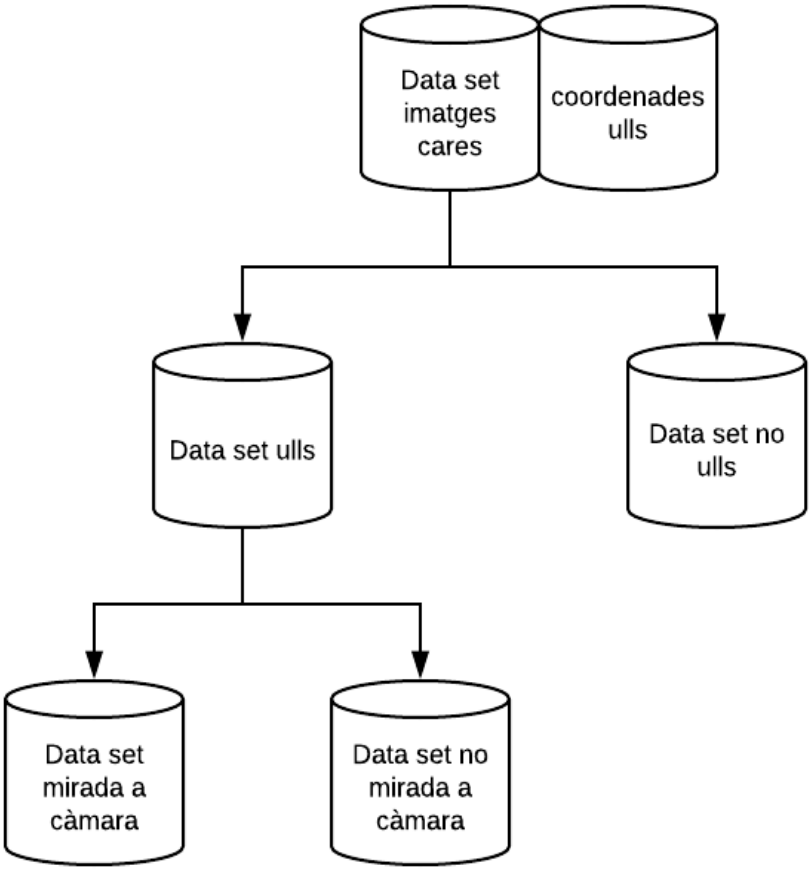
Desenvolupament

Extract data

Primer de tot, partirem d'un gran data set de cares de persones i les coordenades d'on tenen els ulls. A partir d'aquest traurem dos datasets. Un que contindrà ulls i l'altre que contindrà no ulls. Per cada imatge parell d'ulls que trèiem, generarem 20 imatges de no ulls.

Un cop tenim aquests dos datasets ben organitzats, haurem de treure un dataset de mirades. El dataset de ulls el dividirem en dos: mirades i no mirades. Si els ulls miren a la càmera aniran al dataset de mirades i, si no, al de no mirades.

Per tal de generar el dataset no ulls hem fet un algoritme el qual a partir d'una imatge del dataset de cares la retalla random però tenint en compte que mai hi contingui ulls.



Obtenció de features

Un cop organitzats els nostres dataset hem de treure les features de cada imatge per després entrenar el nostre predictor. Primer hem de fer un estudi i pensar quines features ens poden anar bé per tal d'aconseguir una bona precisió en el nostre predictor.

El descriptor de característiques de l'histograma de gradients orientats (HOG) i les característiques LBP (prèviament explicats a la introducció) seran les característiques encarregades escollides.

```
featuresHOG = extractHOGFeatures(I)
```

HOG ens retorna un vector amb totes les característiques de la imatge i un plot el qual no utilitzarem pel predictor.

```
featuresLBP = extractLBPFeatures(I)
```

LBP ens retorna un patró binari local uniforme (LBP) extret de la imatge en escala de grisos.

Aquestes característiques les traurem pels diferents datasets. I les ajuntem. Tindrem les features dels ulls i no ulls per una banda i per l'altre la de mirada a càmera i mirada no a càmera.

Primer vàrem implementar HOG tal i com ens deia l'enunciat de la pràctica. Al realitzar HOG vàrem observar una bona precisió, per part del predictor, però no la desitjada. Després de testejar i buscar diferents característiques com mitjanes del nivell de grisos i característiques de regionProps vàrem observar que al unir HOG i LBP ens donava una molt bona precisió. Com veurem després als resultats de les matrius de confusió.

Split data

Un cop tenim els dos dataset de features els dividim. En el nostre cas els dividim en un rati del 0.8, és a dir, 80% s'anirà a training i l'altre 20% a testing. Això ho farem tant per ulls com mirades per tal d'entrenar els nostres predictors.

Processament data

Convertirem els nostres datasets els qual són structs d'arrays bidimensionals en una matriu i en una array (és el que conté si és ulls o no i mirada o no) per tal de poder de passar-se-la al predictor.

Creació d'un model

Un cop tenim totes les dades necessàries i en el format que ens interessa, el següent que hem de fer és crear el model de predicció i entrenar-lo amb aquestes dades. El classificador que hem escollit és TreeBagger. El motiu de la nostra selecció és que a una altra assignatura (Mineria de Dades) hem utilitzat l'algorisme i ens ha donat bons resultats.

Testing i obtenció dels resultats

Un cop creat i entrenat el model amb les dades del training (80% del total), utilitzem la resta de les dades per tal de comprovar la precisió i rendiment del nostre predictor.

Per tal d'aconseguir-ho, utilitzem la funció *predict* de matlab amb dos paràmetres. Aquests seran, per una banda el nostre model creat anteriorment i, per l'altra la matriu de dades del reservades pel testing. El resultat serà un array de prediccions (1 o 0). Aquest array el compararem amb l'array que anteriorment hem obtingut amb les veritats (també 1 o 0). Aquest procés el fem tant pel model de predicció d'ulls com pel model de predicció de mirades.

Finalment, creem la confusion matrix, de la qual podem obtenir la precisió dels nostres predictors.

Interpretació de les dades

Primer de tot, podem observar, en la següent pàgina, una bona precisió dels resultats. Això pot ser degut a la bona elecció de les característiques i descriptors de les imatges. Una altra observació que podem fer és que l'ús característiques HOG, tot i que sí que milloren la precisió del nostre predictor, no ho fan tant com podríem haver esperat en un principi.

Com era d'esperar, la precisió a l'hora de predir si una imatge pertany a dos ulls és més gran que la precisió a l'hora de detectar si uns ulls estan mirant o no a la càmera.

Tot i això, podem estar satisfets amb els resultats obtinguts.

Confusions Matrix

CONF MATRIX ULLS HOG

6074	6
14	294

Accuracy = 0.99687

CONF MATRIX LOOKS HOG

184	15
36	69

Accuracy = 0.8322

CONF MATRIX ULLS NO HOG

6081	9
18	280

Accuracy = 0.99577

CONF MATRIX LOOKS NO HOG

166	21
40	77

Accuracy = 0.79934

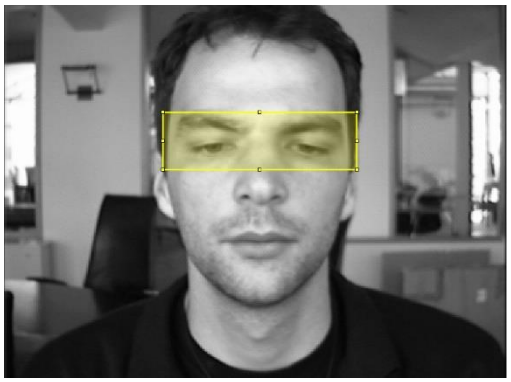
Implementació finestra lliscant

Pel procés del desenvolupament d'una finestra lliscant la qual detecti ulls/no ulls hem utilitzat diferents eines que ens dona matlab, algoritmes propis i els predictors que prèviament hem entrenat.

Per fer una finestra lliscant hem utilitzat una funció del matlab la qual ens implementa una finestra lliscant a sobre d'una imatge i ens retorna les coordenades del rectangle a sobre de la imatge. Utilitzarem aquesta funció per tal de que ens retorni les coordenades i començar a treballar amb la imatge resultant un cop retallada. Utilitzarem un timer per tal de que ens passi el requadre cada x temps, per fer-ho més usable.

```
h = drawrectangle('Color', [1 1 0], 'Position', [2, 2, 120, 40]);
```

```
cropRect = rect.Position;
```



Després d'aconseguir el rectangle retallem la fotografia i obtenim la imatge resultant. Aquesta imatge serà la que haurem de dir si conté ull o no.

```
J = imcrop(I, rect.Position);
```

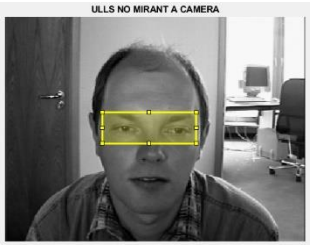
Imatge resultant després de fer el crop amb les dades del rectangle de la finestra lliscant

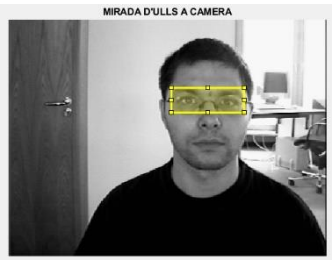
Un cop tenim la imatge sigui d'ull o no li hem de treure les features. La funció que teníem abans per treure les features no ens serveix ja que era per moltes imatges. Hem tingut que fer una nova funció que retorni una matriu sencera però semblant a la anteriorment implementada. Les característiques seran HOG i LBP escollides anteriorment. Li passem a la funció per treure les features la imatge i ens retorna una matriu amb totes les característiques de la fotografia. Aquestes característiques se les passarem al predictor.

Enviem les característiques el predictor per detectar si son ulls o no. En el cas de que ens detecti ulls li passarem al segon predictor per tal de que determini si estan mirant a la càmera o no.

Resultats

Com hem vist anteriorment observem una bona accuracy en els resultats. Podem dir que els nostres predictors en la teoria funcionen raonablement bé. Al desenvolupar la finestra lliscant i veure diferents resultats observem també una molt bona precisió en la detecció de ulls/no ulls i una pitjor precisió en la detecció de la mirada. A continuació podem observar els diferents resultats dels nostres predictors amb la implementació de la nostre finestra lliscant.





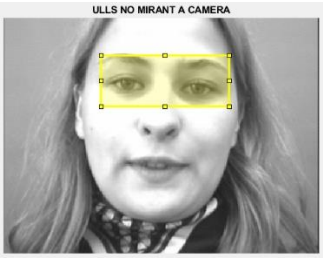
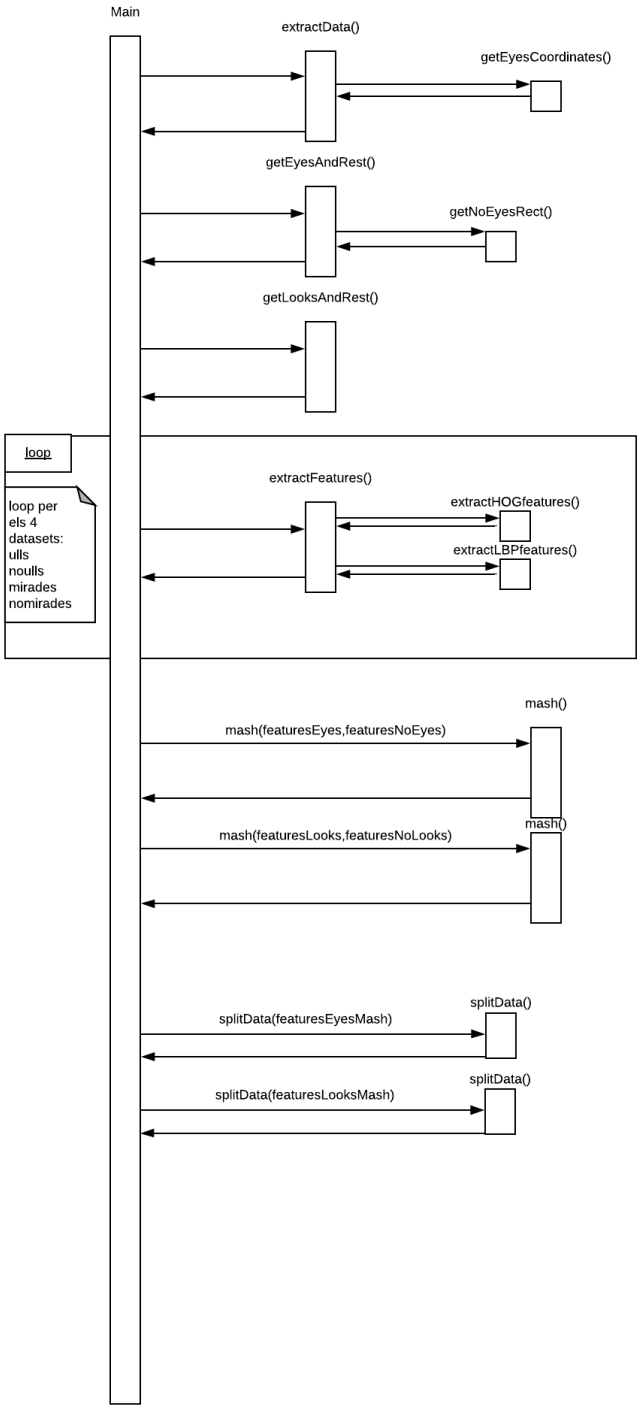
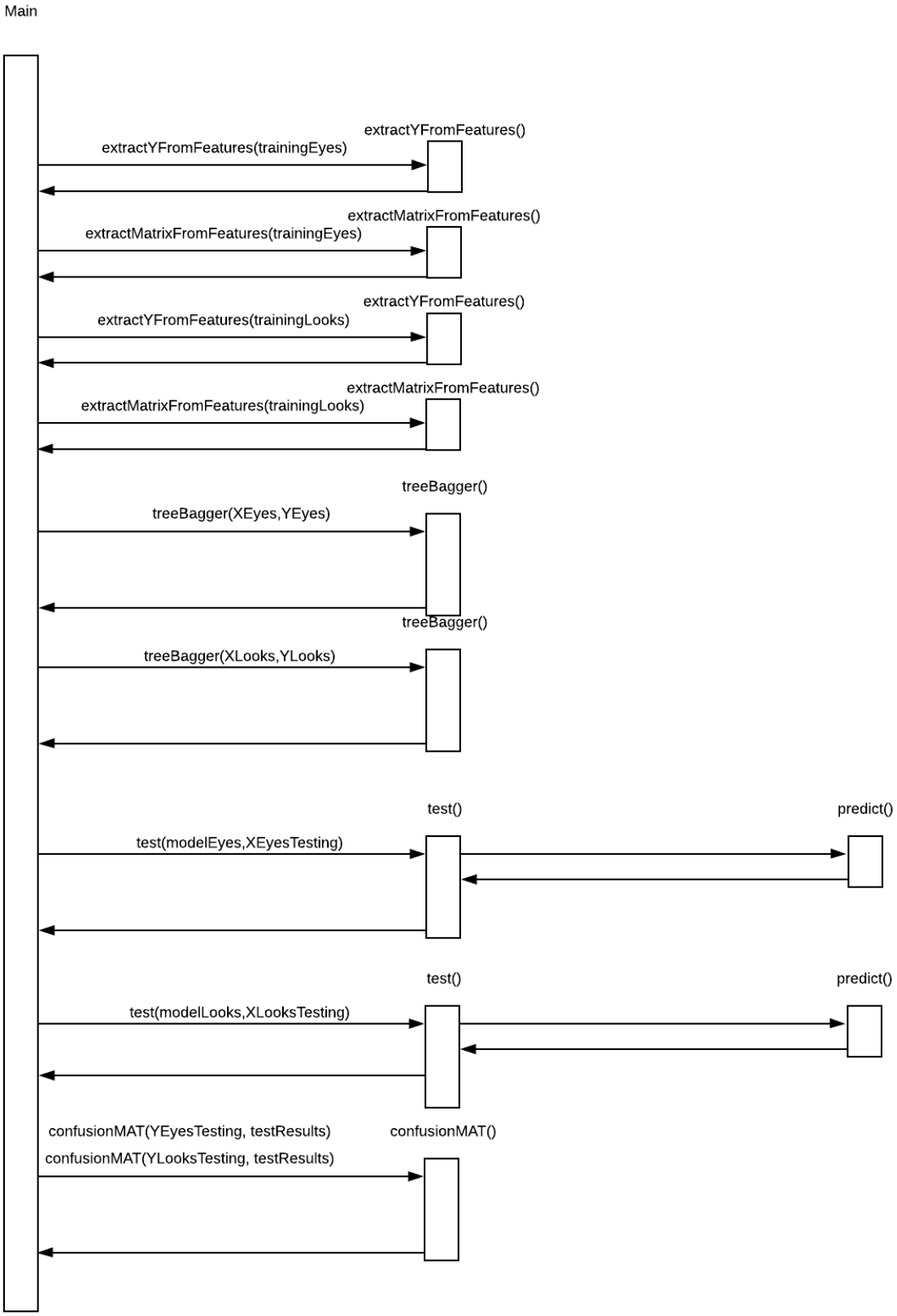


DIAGRAMA UML DEL PROGRAM




Main
extractYFromFeatures()
extractYFromFeatures(trainingEyes)
extractMatrixFromFeatures()
extractMatrixFromFeatures(trainingEyes)
extractYFromFeatures()
extractYFromFeatures(trainingLooks)
extractMatrixFromFeatures()
extractMatrixFromFeatures(trainingLooks)
treeBagger()
treeBagger(XEyes,YEyes)
treeBagger()
treeBagger(XLooks,YLooks)
test()
predict()
test(modelEyes,XEyesTesting)
test()
predict()
test(modelLooks,XLooksTesting)
confusionMAT(YEyesTesting, testResults)
confusionMAT()
confusionMAT(YLooksTesting, testResults)