Detecció de quadres des d'un dispositiu mòbil

PFC Visió per Computador

Jordi Miquel Oliver Buades
05/06/2013

Índex

Introducció

Actualment, els dispositius mòbils són cada cop un element més freqüent a la vida de les persones. En especial, cal destacar el creixement en els últims anys de la venda i ús del que s'anomenen *Smartphones* o telèfons intel·ligents. En conseqüència, i també degut a les constants millores en quan a capacitat de processament que van sofrint aquests dispositius, s'ha obert un gran ventall de possibilitats en torn a les aplicacions que poden executar. Al mercat de les aplicacions mòbils, podem trobar exemples d'aplicacions que s'aprofiten d'aquesta evolució i que van des de la captura i edició de vídeo i fotografies en alta resolució, fins a jocs amb gràfics realment espectaculars.

En aquest projecte, l'objectiu és precisament aprofitar la potència que ens ofereixen els dispositius actuals, per tal de crear una aplicació mòbil capaç d'identificar quadres i parts dels quadres en temps real. Concretament, s'ha desenvolupat l'aplicació per a dispositius que empren el sistema operatiu *iOS*, tot i que una aplicació d'aquest estil es podria portar a altres sistemes operatius mòbils com *Android* o *Windows Phone*.

A grans trets, per tal de portar a terme la identificació d'un quadre o de la part d'un quadre, necessitem emprar un algorisme que tingui la capacitat d'identificar les característiques de la imatge que es captura amb el dispositiu, i comparar-les a les d'una imatge de la qual n'haurem extret les característiques prèviament, durant el que anomenarem fase d'entrenament. En cas de que coincideixin prou característiques entre la imatge capturada i la que hem emprat a la fase d'entrenament, podrem considerar que hem detectat la imatge. Tot aquest procediment no és senzill d'implementar, i donat el temps limitat per a la realització del projecte, s'ha escollit emprar la implementació existent de l'algorisme *Random Ferns* a les llibreries *OpenCV*.

Tot i que com ja s'ha dit, els dispositius mòbils han sofert una gran evolució en quan a capacitat de processament, aquest procés que hem anomenat fase d'entrenament és un procés massa pesat en quan a cost computacional per a portar-lo a terme amb una aplicació mòbil. Per tant, en aquest projecte també s'ha desenvolupat una aplicació nativa per al sistema operatiu *OS X*, capaç d'entrenar l'algorisme *Random Ferns* per a la detecció d'una imatge en concret. En particular, aquesta aplicació dóna la possibilitat de generar un model, que després pot ser carregat a l'aplicació mòbil desenvolupada, i que conté la informació necessària per a detectar una imatge o les seves parts. Com a aspecte innovador en aquesta aplicació d'entrenament, les parts d'un quadre, així com la seva descripció, es poden obtenir dels articles de la *Wikipèdia* que parlen d'aquesta imatge. Per exemple, si volem generar un model amb la capacitat de reconèixer el quadre de *Las Meninas* i alguns dels personatges que hi apareixen, serà suficient amb introduir la *URL* de l'article que parla sobre el quadre, i seleccionar de la llista

d'imatges de l'article que es mostren, quines ens interessa detectar com a parts del quadre.

Reconeixement de característiques d'una imatge emprant Random Ferns

Com ja s'ha comentat a la introducció, el procés d'extreure les millors característiques d'una imatge per a la seva posterior identificació és, especialment en termes computacionals, un procés pesat. Amés, s'ha de tenir en compte que perquè la detecció sigui el més robusta possible, els canvis en l'escala o rotació de la imatge capturada respecte a la imatge d'entrenament ens haurien d'afectar el mínim possible. D'altra banda, la detecció haurà d'esser el suficientment ràpida, ja que aquest procés s'executarà al dispositiu mòbil amb l'objectiu que es porti a terme en temps real, emprant la càmera del dispositiu.

Per assolir aquest compromís entre robustesa i rapidesa, s'ha escollit l'algorisme *Random Ferns*, ja que com es descriurà a continuació, ens proporciona aquestes característiques.

La fase d'entrenament

L'objectiu d'aquesta fase, tal com ja s'ha explicat, és la de generar un model capaç de reconèixer una imatge de forma robusta i ràpida emprant *Random Ferns*.

Per començar el procés d'entrenament amb *Random Ferns*, en primer lloc haurem de detectar els punts clau més estables de la imatge que emprarem per a generar el model, aplicant-li un nombre determinat de transformacions. A cadascuna d'aquestes transformacions, es seleccionen una sèrie de punts clau. Un cop acaba aquest procés, es consideraran els punts clau més estables aquells que hagin aparegut de forma més freqüent durant tota la sèrie de transformacions portades a terme, i se'ls hi assignarà una classe determinada a cadascun d'ells.

El següent pas consisteix en generar un conjunt d'entrenament suficientment robust per a cadascuna de les classes obtingudes al pas anterior. En aquest punt apareix el concepte de *patch*, que és la zona de la imatge que envolta un determinat punt clau. Per generar el conjunt d'entrenament de cadascuna d'aquestes classes, s'apliquen una sèrie de transformacions a cada *patch* que inclouen desenfocament gaussià i rotacions, i que faran més efectiva la detecció enfront al renou i l'angle de la imatge capturada. Com més gran sigui aquest conjunt d'entrenament, més efectiva serà la detecció.

Com a resultat de l'entrenament de cada classe, es genera per a cadascuna d'elles un nombre determinat de *Ferns*, que denotarem com M. Un *Fern* és un conjunt de característiques binaries que té una mida específica, que denotarem com S. Per tant cada classe tindrà associada M x S característiques binaries, que ens serviran durant la fase

de detecció per tal de determinar a quina classe, si es que en correspon a alguna, pertany un punt clau específic de la imatge capturada. Cal destacar que si el nombre de *ferns* o la mida d'aquests són valors molt grans, la mida en memòria del model generat també serà molt gran, ja que cada *Fern* pot prendre 2^S valors. Suposant un cas on tenim 200 classes i una quantitat de 30 *Ferns* de mida 10 associats a cadascuna d'elles, estaríem parlant d'un model que identifica una imatge i que té una mida de $200 * 30 * 2^{10} * 4 = 24576000$ bytes $\approx$ 23,5 MB. Treballant amb un dispositiu mòbil actual, pot ser no sembla un nombre massa gran, però de fet ho és ja que la intenció en aquest projecte és la de generar models que no solament siguin capaços d'identificar una sola imatge, sinó també diverses parts de cada imatge. Per tal de mitigar aquest problema, s'ha emprat compressió per tal de fer que el fitxer d'un model ocupi el mínim possible.

La fase de detecció

A la fase de detecció el que volem és saber quants punts clau de la imatge que hem capturat coincideixen amb el model que hem generat. Per fer-ho, s'hauran d'extreure N punts clau de la imatge capturada. En aquest aspecte, la diferència respecte la fase d'entrenament és que aquests punts claus no són els més estables. Extreure els punts claus més estables requereix massa temps si el que es pretén és una detecció en temps real. En aquesta fase s'extreuen molts més punts que a la fase anterior, d'aquesta manera, la probabilitat que algun d'aquests punts extrets coincideixi amb alguna de les classes del model generat a la fase d'entrenament, és major. La classificació es portarà a terme aplicant els tests binaris del model, sobre cadascun dels punts claus extrets, per determinar a quina classe pertanyen en cas que en pertanyin a alguna.

Un cop vist quants punts hem aconseguit trobar del model a la imatge capturada, tindrem un valor que ens indicarà el percentatge de detecció, que ve donat per el *Nombre de punts clau identificats / Nombre de punts clau del model*, on el *Nombre de punts clau identificats* és menor o igual que el *Nombre de punts clau del model*.

Disseny de les aplicacions

Ja s'ha comentat que la generació dels models es portarà a terme amb una aplicació d'escriptori, en aquest cas implementada per al sistema operatiu *OSX*, però que seria fàcilment portable a altres plataformes. El mateix passa amb l'aplicació mòbil, la qual s'ha portat a terme per a dispositius *iOS*.

En aquest capítol, es presenta el disseny de cadascuna d'aquestes aplicacions, en especial el de les funcions més importants tals com són l'entrenament del model capaç de detectar un quadre i les seves parts, així com la pròpia detecció. Començarem per tant presentant el diagrama de casos d'ús complet, on es denoten quines són les funcionalitats de cadascuna de les aplicacions així com els actors que intervenen en el seu ús.

Diagrama de casos d'ús

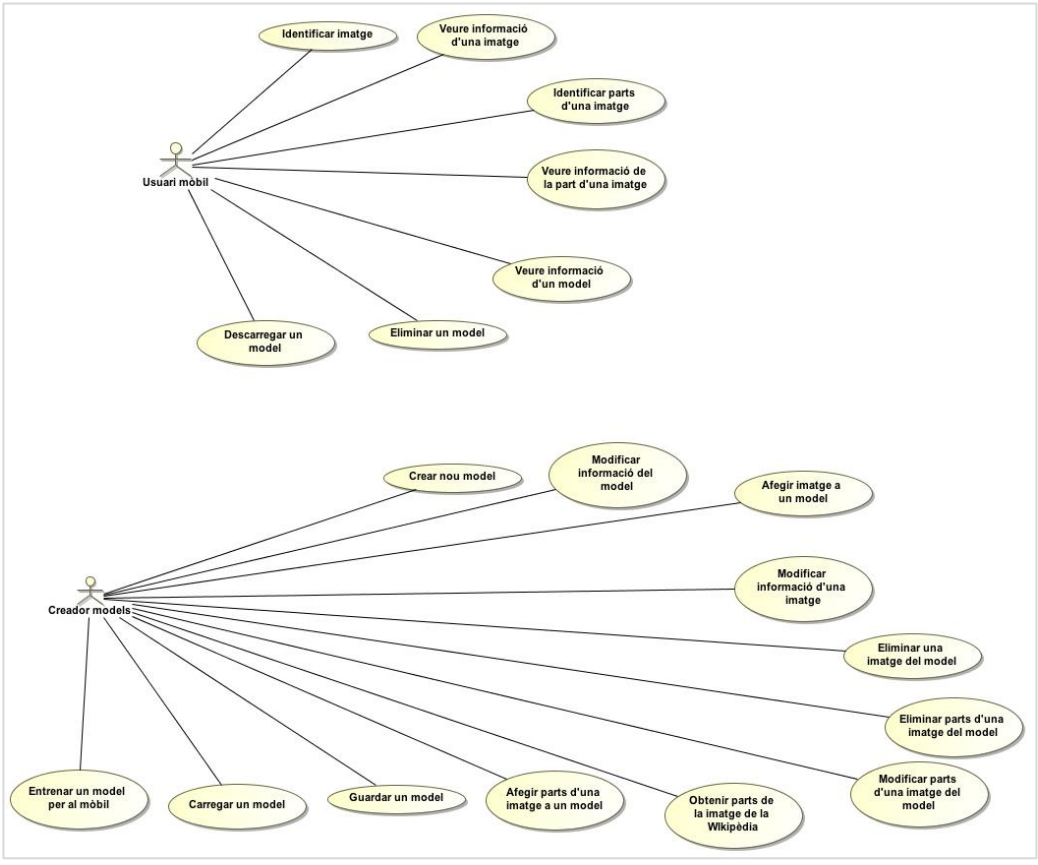

Figura 1. Diagrama de casos d'ús de l'aplicació d'entrenament i de l'aplicació mòbil

Resumint el diagrama, veiem que s'ha de desenvolupar una aplicació d'entrenament que sigui capaç del següent:

- Crear un nou model

- Guardar un model
- Carregar un model
- Modificar la informació d'un model existent. Com a informació bàsica del model que es podrà editar tindrem el nom del model.
- Afegir una imatge a un model
- Modificar la informació de la imatge. Concretament aquí es fa referència al títol del quadre, a l'autor i a l'any de la que data l'obra.
- Eliminar una imatge del model
- Obtenir parts del quadre a partir de l'enllaç a un article de la *Wikipèdia*.
- Afegir les parts seleccionades al model.
- Modificar les parts afegides d'una imatge, donant la possibilitat d'editar-ne el títol de la part així com la seva descripció.
- Eliminar parts afegides a una imatge del model.

Un punt a tenir en compte és que es treballarà amb dos tipus de model: d'una banda, un model que contindrà les diferents imatges així com les parts de cadascuna, amés de la seva informació associada; d'altra banda, quan es generi el model consumible per a l'aplicació mòbil, serà quan es porti a terme l'entrenament dels diferents detectors de cada imatge. Amb aquesta separació, aconseguirem que un usuari pugui treballar sobre un model durant un temps determinat, afegint-hi imatges, editant-ne la informació, etc. i que posteriorment, un cop el tingui enllestit, sigui capaç d'exportar-lo a la versió que es podrà utilitzar amb l'aplicació mòbil.

Pel que fa a l'aplicació mòbil, tindrà les següents funcionalitats:

- Identificar una imatge.
- Veure la informació de la imatge identificada.
- Identificar les parts d'una imatge.
- Veure la informació de les parts identificades de la imatge.
- Veure la informació dels models carregats. Això inclou veure quines imatges té cadascun dels models, juntament amb la informació de cada imatge, i també la possibilitat de visualitzar les parts que es poden identificar de cada imatge així com la seva informació associada.
- Eliminar un model
- Carregar un model.

En aquest cas, cal destacar especialment la càrrega del model. Aquesta es portarà a terme des d'una URL on estarà penjat el model. D'aquesta manera, qualsevol persona que sigui capaç de generar un model podrà penjar-lo a *Internet*, permetent a usuaris de l'aplicació mòbil descarregar-lo.

Disseny de l'aplicació d'entrenament

L'aplicació d'entrenament és el component clau per tal d'aconseguir efectivitat en la detecció a l'aplicació mòbil. Per aquest motiu, s'ha pensat en una estructura de classes que en un futur sigui fàcilment ampliable, i que sigui fàcil de mantenir. Per tal d'assolir aquest compromís, l'ús de patrons de disseny ha estat clau, en especial l'ús del patró *delegate*, molt estès en plataformes *OSX* i *iOS*. Aquest patró, tal com el seu nom indica, ens permet delegar certa funcionalitat d'una classe a una altra classe associada. Durant tot el projecte, s'empra principalment amb l'objectiu de notificar a diferents controladors quan ha acabat una tasca, de manera que aquests últims tinguin la responsabilitat de portar a terme les accions que creguin oportunes.

Model de dades

A la figura 2, es presenta el model de dades de l'aplicació d'entrenament. Es pot observar que el que anomenem *Model* és, en essència, el conjunt d'imatges i parts d'aquestes imatges que voldrem detectar en un futur al dispositiu mòbil.

També es pot apreciar com es fa ús de la classe pròpia *NSImage* del *Cocoa Framework* d'*OSX*. Aquesta classe és la representació en memòria d'una imatge, la qual emprarem per tal d'entrenar un detector capaç d'identificar-la posteriorment.

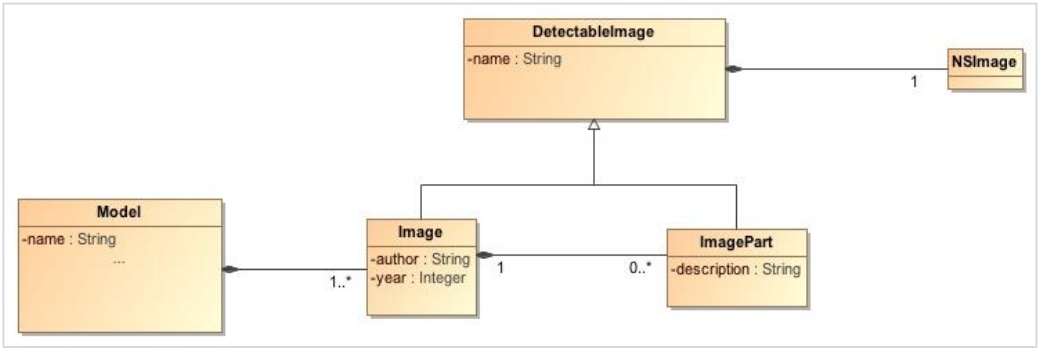

Figura 2. Model de dades de l'aplicació d'entrenament.

Disseny de les pantalles

Un cop especificada la funcionalitat de l'aplicació d'entrenament al diagrama de casos d'ús, cal definir les pantalles de l'aplicació on podrem realitzar les diverses accions definides.

Comencem amb la pantalla principal de l'aplicació d'entrenament. Des d'aquí, es podran portar a terme les següents accions:

- Crear un nou model.
- Guardar un model.
- Carregar un model.
- Afegir una imatge al model.
- Modificar la informació d'una imatge d'un model o de la part d'una imatge.
- Eliminar una imatge del model o eliminar la part d'una imatge.
- Exportar el model a la versió per a mòbil.

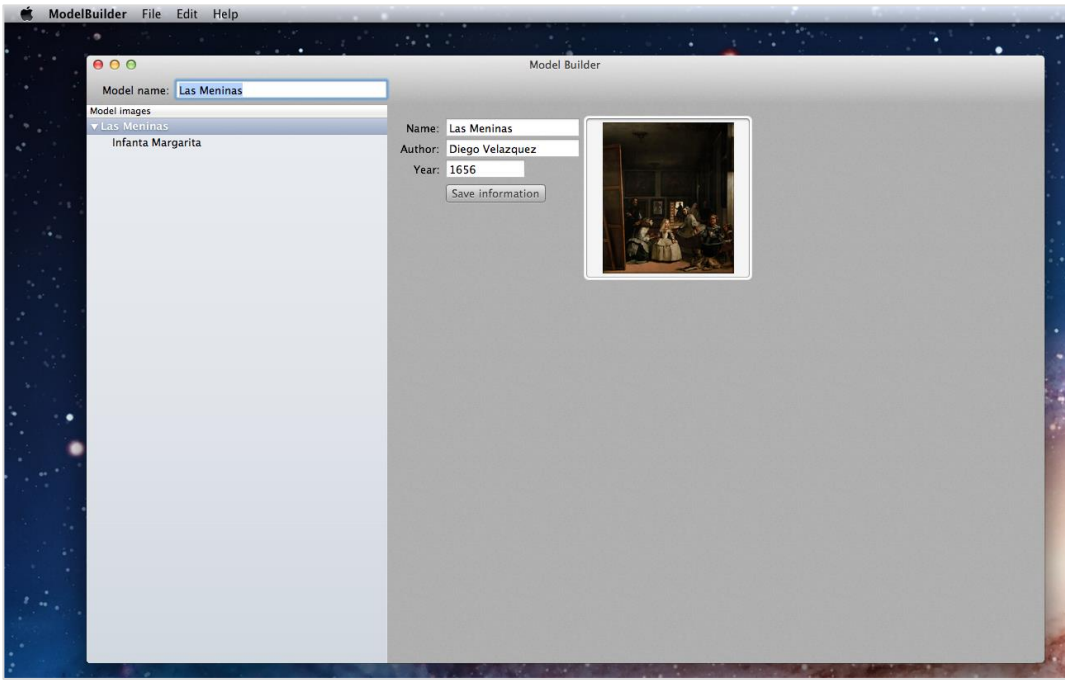

Figura 3. Pantalla principal de l'aplicació d'entrenament.

Com es pot observar a la figura 3, a un costat tenim la llista d'imatges que composen el model. Per a cadascuna de les imatges, també tenim la llista de parts. A la

part dreta de la finestra, tenim la possibilitat d'editar la informació de les imatges i les parts. A la barra superior, podem editar de forma ràpida el nom del model. Finalment, la resta d'opcions esmentades anteriorment les podem trobar als menús superiors *File* i *Edit*.

La següent pantalla que es mostra a la figura 4, és la que ens permetrà obtenir les diferents parts d'una imatge arrel d'un article de la *Wikipèdia*, així com la descripció de cada part des del peu de foto de la imatge. Un cop descarregades les diferents fotos amb la seva descripció, tindrem la possibilitat de triar quines formen part del quadre que volem identificar, per tal de que siguin identificades sobre l'obra.

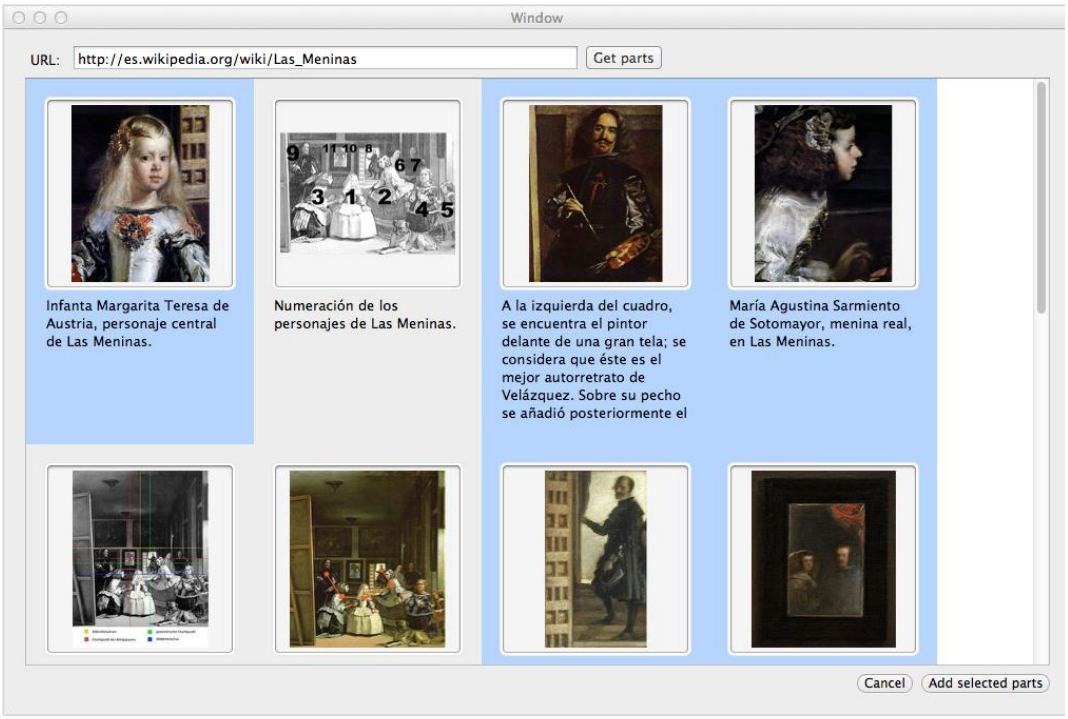

Figura 4. Pantalla de descàrrega de parts de la imatge des d'un article de la *Wikipèdia*. Es pot observar que en blau hi ha marcades les diferents parts que ens interessa afegir a la imatge que volem identificar.

Diagrama de classes

El diagrama de classes de l'aplicació d'entrenament està dividit en cinc mòduls o paquets diferents. Tres d'ells són corresponents al patró de disseny MVC (Model Vista Controlador). Els altres dos, representen components encarregats de l'entrenament i de la obtenció de parts de les imatges des de la *Wikipèdia*.

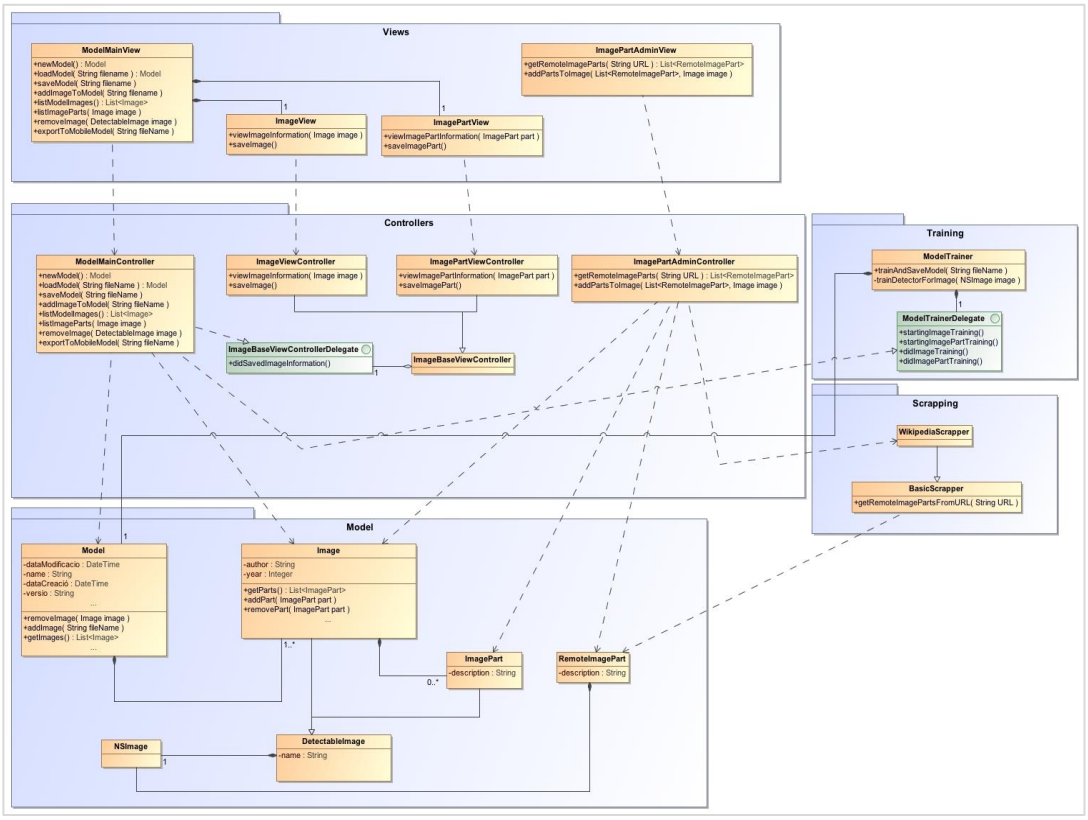

Figura 5. Diagrama de classes de l'aplicació d'entrenament.

Com es pot observar, l'aplicació està composta essencialment per les vistes *ModelMainView* i *ImagePartAdminView*. La primera d'elles té alhora dues vistes filles que tenen com a objectiu la visualització i edició de la informació d'una imatge, així com la visualització i edició de la informació de la part d'una imatge, respectivament.

Cadascuna de les vistes o vistes filles té alhora associat un únic controlador. En aquest controlador es proporciona la funcionalitat bàsica de cadascuna de les vistes. Si observem el diagrama, podem veure un primer ús del patró *delegate*: la vista principal (*ModelMainView*), implementa la interfície *ImageBaseViewControllerDelegate*, de forma que els controladors encarregats de visualitzar i editar la informació de les imatges i les parts, delegaran la responsabilitat de les accions a realitzar al controlador *ModelMainView* un cop hagi finalitzat el guardat de la informació d'un element del model.

Diagrames de seqüència

Fins ara, hem vist quines son les funcionalitats bàsiques de l'aplicació d'entrenament, el disseny de les pantalles i el disseny de les diferents classes. Per tal d'aprofundir encara més en el disseny de l'aplicació, mostrarem i explicarem els diagrames de seqüència corresponents als casos d'ús més rellevants on es porten a terme operacions amb el model.

El primer diagrama que analitzarem serà el referent al cas d'ús "Afegir imatge a un model". Com es pot observar a la figura 6, l'usuari encarregat de crear el model és qui hi afegeix la imatge des d'un fitxer. A partir d'aquí, el controlador principal s'encarrega de dir-li al model que ha d'afegir una imatge.

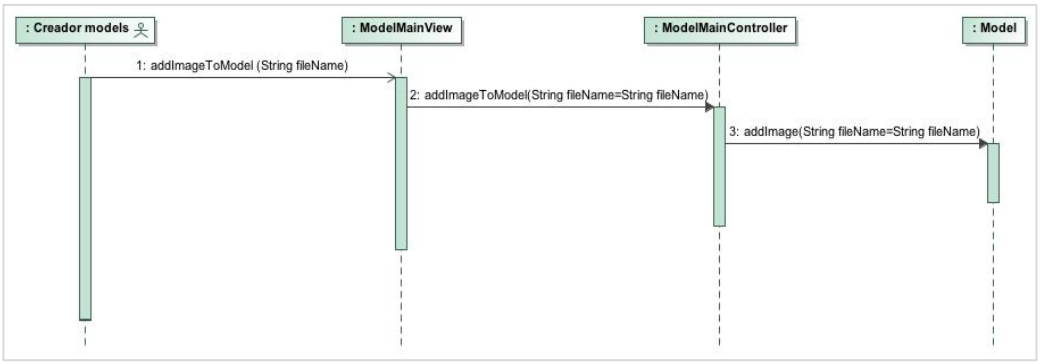

Figura 6. Diagrama de seqüència corresponent al cas d'ús "Afegir imatge a un model".

Com ja hem comentat, un cop es descarreguin les imatges relacionades amb un article de la *Wikipèdia*, tindrem la possibilitat de seleccionar les que ens interessin per tal d'afegir-les a una imatge que ja tenim al model. Aquest últim pas, queda representat en el diagrama de seqüència de la figura 7.

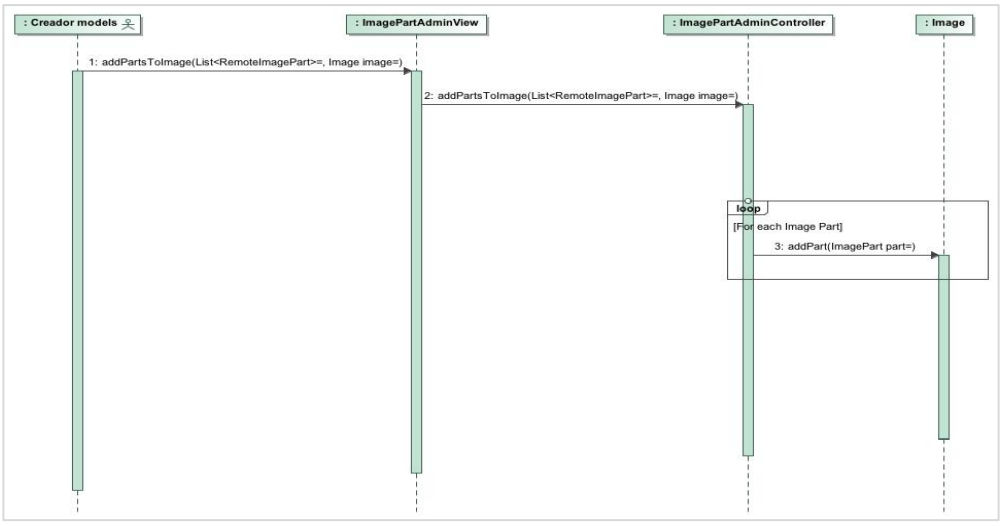

Figura 7. Diagrama de seqüència corresponent al cas d'ús "Afegir parts d'una imatge a un model".

En relació al diagrama de la figura 7, hem parlat del fet d'obtenir les parts d'una imatge a partir d'un article de la *Wikipèdia*. Al diagrama de la figura 8, es presenta aquest procés, on l'únic que necessitem és la *URL* de l'article.

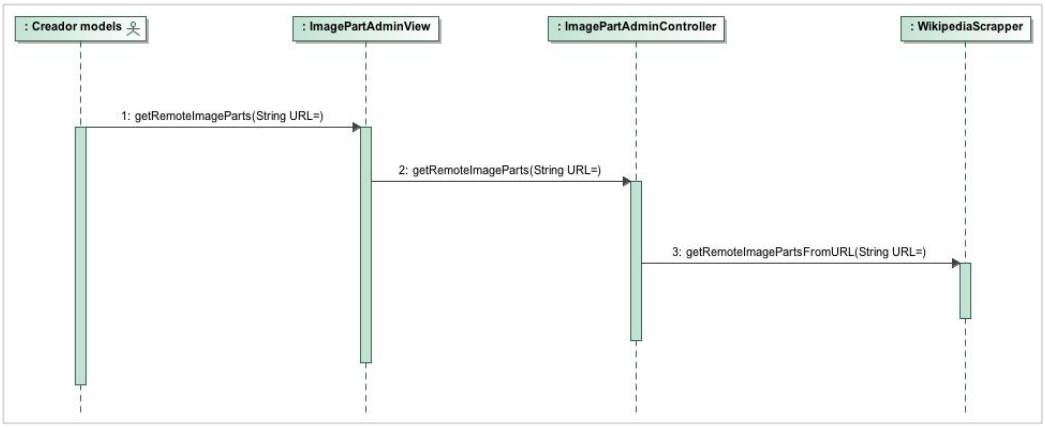

Figura 8. Diagrama de seqüència corresponent al cas d'ús "Obtenir parts d'una imatge de la Wikièdia".

L'altra funcionalitat imprescindible que cal representar, és l'entrenament d'un model capaç de ser llegit per l'aplicació mòbil. Al diagrama de seqüència de la figura 9, es pot observar com s'empra el patró de disseny *delegate* per tal de notificar al controlador de les accions que va portant a terme la classe *ModelTrainer*.

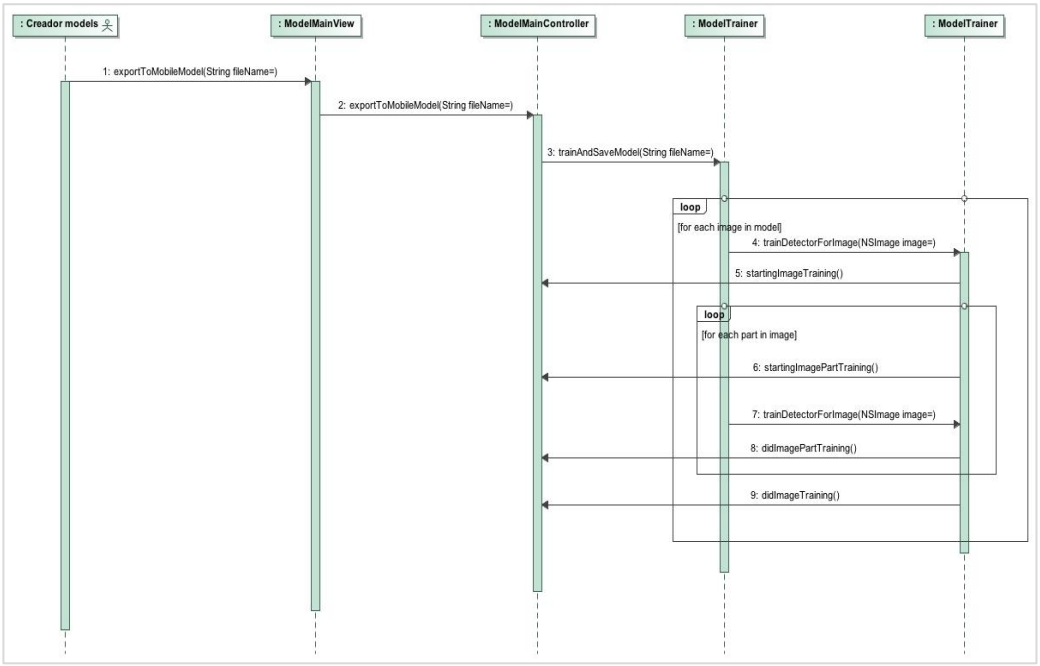

Figura 9. Diagrama de seqüència corresponent al cas d'ús "Entrenar un model per al mòbil".

Els diagrames que queden per presentar, són relatius a les operacions de creació d'un nou model, càrrega a l'aplicació d'entrenament d'un model existent, eliminació d'imatges i eliminació de parts d'imatges. No es tracta de funcionalitats amb gaire complicació, ni presenten cap aspecte destacable, ja que els mateixos diagrames són bastant explicatius.

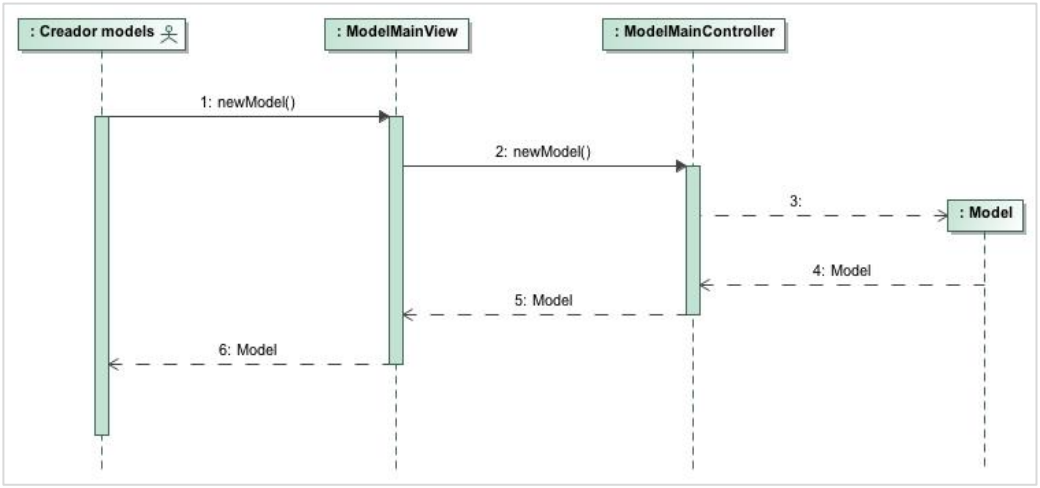

Figura 10. Diagrama de seqüència corresponent al cas d'ús "Crear un nou model".

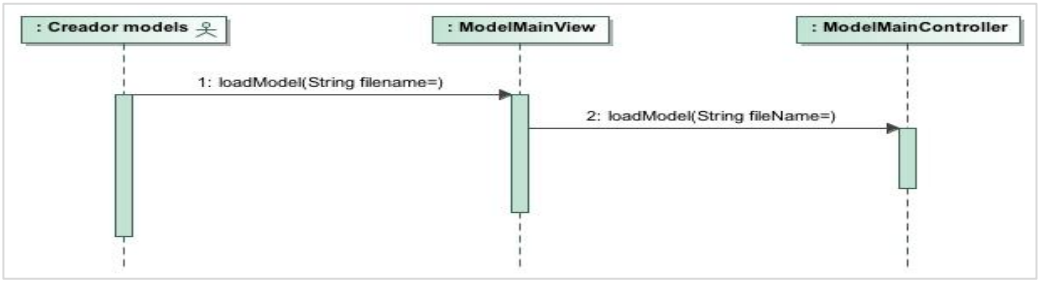

Figura 11. Diagrama de seqüència corresponent al cas d'ús "Carregar un model".

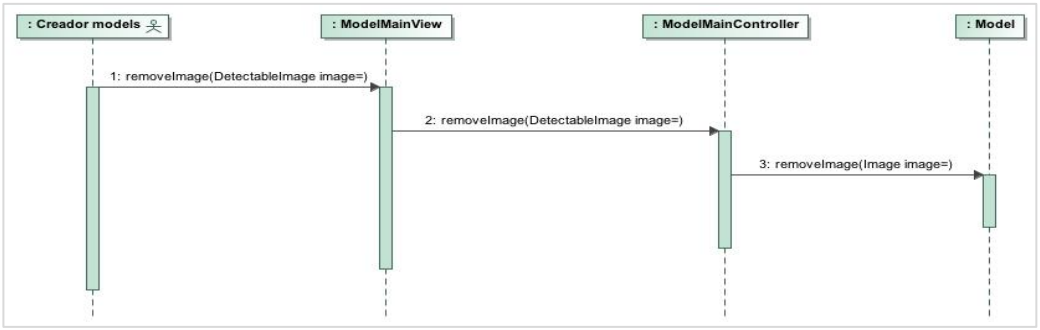

Figura 12. Diagrama de seqüència corresponent al cas d'ús "Eliminar imatge d'un model".

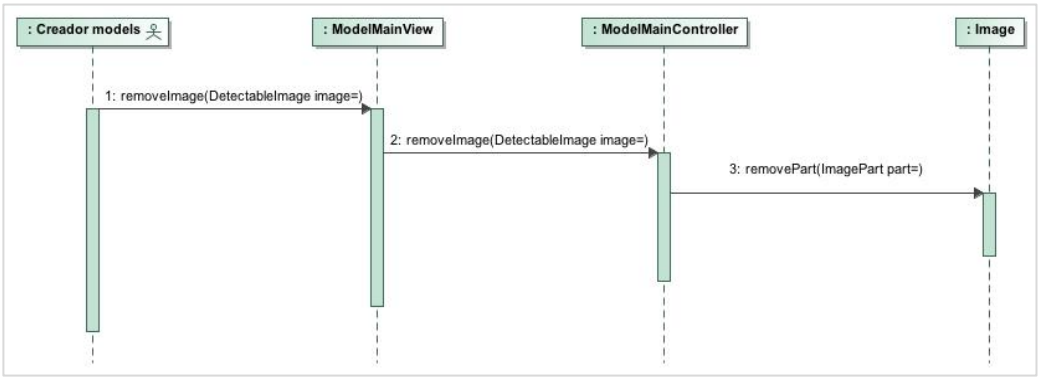

Figura 13. Diagrama de seqüència corresponent al cas d'ús "Eliminar parts d'una imatge del model".

Disseny de l'aplicació mòbil

L'aplicació mòbil representa l'objectiu principal d'aquest projecte: el reconeixement d'obres d'art en temps real a través de la càmera del dispositiu. Tot i que l'aplicació té una estructura senzilla, ja que consta d'una vista per a capturar el vídeo i d'altres per a gestionar els models carregats al dispositiu, s'ha de tenir en compte que un mòbil és bastant limitat en quan a característiques de processament enfront d'un ordinador. Per tant, per tal de treballar amb un o més models que puguin contenir varies imatges cadascun, s'hauran d'aprofitar al màxim les opcions de paral·lelisme que ens ofereix l'SDK del dispositiu, en aquest cas *iOS*, de forma que els recursos s'aprofitin de la forma més òptima possible. Això fa que aquest disseny que es presenta estigui pensat tenint en compte la plataforma concreta on s'executarà l'aplicació. Tot i això, un disseny per a dispositius *Android* o *Windows Phone* seria molt similar, ja que tant *Java* com *.Net* ofereixen característiques semblants a *Objective C* pel que fa a la gestió de *threads* i tasques.

Tal com hem fet amb l'aplicació d'entrenament, començarem presentant el disseny de les pantalles, i continuarem amb la presentació dels diagrames de classes i de seqüència.

Model de dades

El primer de tot que podem apreciar és que el model de dades de l'aplicació mòbil és molt semblant al de l'aplicació d'entrenament. La única diferència és el fet de que ja no tenim un objecte de tipus *NSImage* relacionat amb els objectes de tipus *DetectableImage*. El principal motiu d'aquest fet és que podria resultar massa pesat traslladar al mòbil cadascuna de les imatges emprades durant l'entrenament. Per contra, tenim associat als objectes de tipus *DetectableImage* un objecte de tipus *PlanarObjectDetector*. Es tracta d'un objecte propi d'*OpenCV*, que es genera a la fase d'entrenament i que representa el classificador capaç d'identificar la imatge a la que està associada.

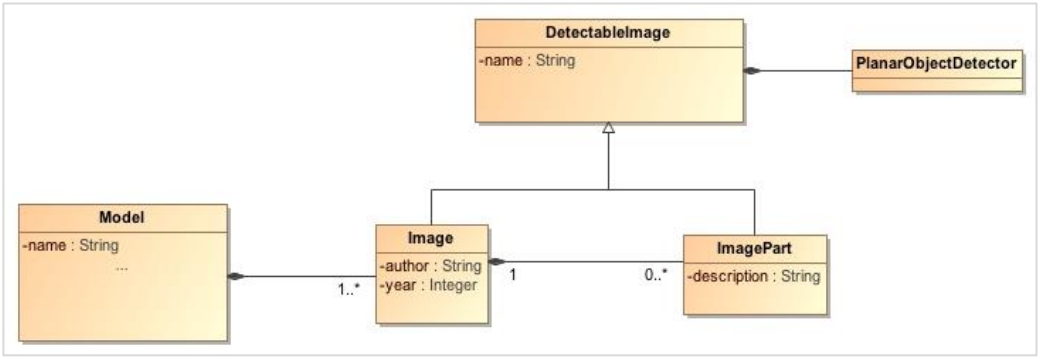

Figura 14. Model de dades de l'aplicació mòbil.

Disseny de les pantalles

La primera pantalla que presentarem es la de captura. Aquesta pantalla és prou senzilla, ja que simplement mostra la imatge que es captura a traves de la càmera del dispositiu, i en cas de detectar-la o detectar-ne les parts, ho indica a sobre de la imatge.

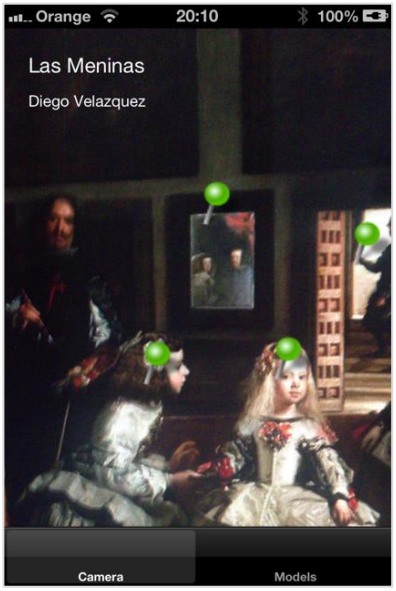

Figura 15. Pantalla de captura de la imatge

A la figura 15, es pot observar com queden marcades les parts de la imatge detectada amb un *pin* de color verd. En cas de tocar un d'aquests *pins*, apareix una finestra modal al dispositiu que ens mostra la informació de la part en qüestió, com podem observar a la figura 16.

Figura 16. Pantalla on es mostra la informació d'una de les parts detectades.

Com ja s'ha comentat, els models els podrem descarregar a l'aplicació mòbil des d'una *URL* on l'usuari que els ha creat els haurà penjat prèviament. Una barra de progrés ens mostrarà l'estat del procés de descàrrega del model al dispositiu mòbil, com es pot apreciar a la figura 17.

Els diferents models que anem descarregant al dispositiu els podrem gestionar a traves de la pantalla que es mostra a la figura 18. Des d'ella, podrem afegir un altre model, el que ens portarà a la pantalla de la figura 17, o podrem eliminar algun dels models que hem carregat. A més, tindrem la possibilitat de veure quines imatges pot reconèixer cada model, i quines parts es poden identificar de cadascuna de les imatges, com es pot veure a les figures 19 i 20.

Figura 17. Pantalla que ens permet afegir un model des d'una *URL*.

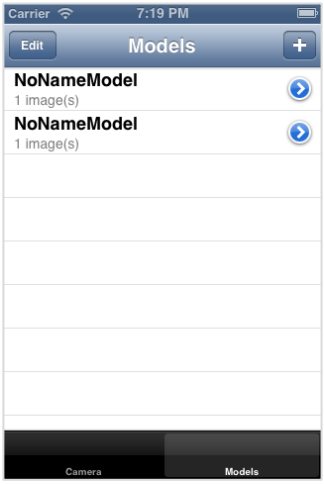

Figura 18. Pantalla que ens permet gestionar els models.

Figura 19. Pantalla on es mostra la llista de parts que es poden reconèixer d'una imatge determinada.

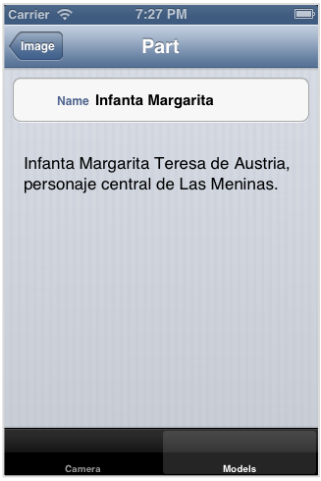

Figura 20. Pantalla on es mostra la informació bàsica d'una part identificable d'una imatge.

Diagrama de classes

El diagrama de classes de l'aplicació mòbil està dividit amb cinc mòduls o paquets diferents. Igual que a l'aplicació d'entrenament, també es segueix el patró *MVC* (Model-Vista-Controlador) per al disseny de l'aplicació. Amés, s'han especificat dos paquets més: el primer, anomenat *Detection*, conté el relatiu a la gestió dels models carregats al dispositiu mòbil així com a la detecció d'imatges i parts. El segon, *OpenCV*, es mostra per claredat, ja que en ell mostrem les classes d'aquesta llibreria que emprem per a gestionar la càmera del dispositiu. Un altre dels objectes d'aquesta llibreria, el *PlanarObjectDetector*, apareix al paquet model degut a la forta relació que existeix amb la classe *DetectableImage*, ja que totes les imatges del model tenen un detector associat i entrenat per a detectar la imatge en qüestió. Els objectes de tipus *Mat* que es passen per paràmetre a les funcions relacionades amb la detecció, també pertanyen a la llibreria *OpenCV*.

D'igual forma que amb l'aplicació d'entrenament, es pot observar al diagrama com el patró *delegate* també és present el desenvolupament d'aplicacions *iOS*. En aquest cas, l'emprem especialment per a notificar al controlador de la vista de captura quan s'ha detectat una imatge i quan s'ha detectat la part d'una imatge. D'aquesta manera és responsabilitat del controlador actuar en conseqüència en front a les dues situacions.

Com a últim punt destacable, cal esmentar que la detecció de les imatges es porta a terme mitjançant *operations*. Es tracta d'un mecanisme que ens ofereix *Objective C* que ens permet enviar operacions a una cua d'operacions. El propi sistema operatiu s'encarrega d'executar-les amb un grau màxim de paral·lelisme, abstraient al programador de gestionar els fils que intervenen en el procés. Nosaltres hem definit una operació pròpia, que té associada una imatge detectable, i que s'encarrega d'executar el mètode de detecció. Quan la operació acaba amb el procés, notifica al controlador del model, informant-lo d'aquest fet i passant-li la imatge detectada així com els punts que representen la zona rectangular on està situada la imatge respecte del *frame* capturat.

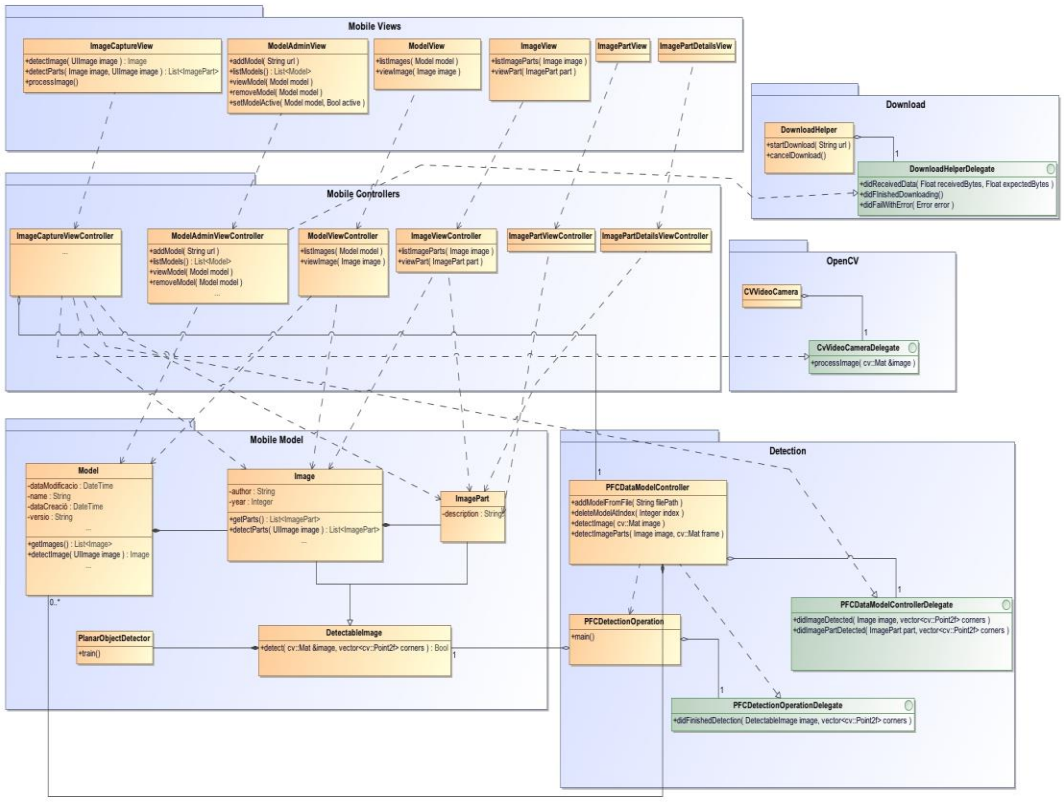

Figura 21. Diagrama de classes de l'aplicació mòbil.

Diagrames de seqüència

A continuació, es presenten els diagrames de seqüència de l'aplicació mòbil pel que fa a la detecció. Es pot observar el que passa a l'aplicació des del moment en que es captura un *frame* a traves de la càmera del dispositiu, fins que es notifica al controlador de la vista de captura que s'ha detectat una imatge.

Les dues seqüències son pràcticament idèntiques, ja que el procés per a detectar una imatge o la part d'una imatge és pràcticament el mateix. Varia especialment en el fet de notificar al controlador de captura si el que s'ha detectat és una imatge, o per contra es la part d'una imatge. La figura 22, representa el procés on s'intenta identificar una imatge donada la captura d'un *frame*, mentre que la figura 23, representa el mateix procés però per la part d'una imatge.

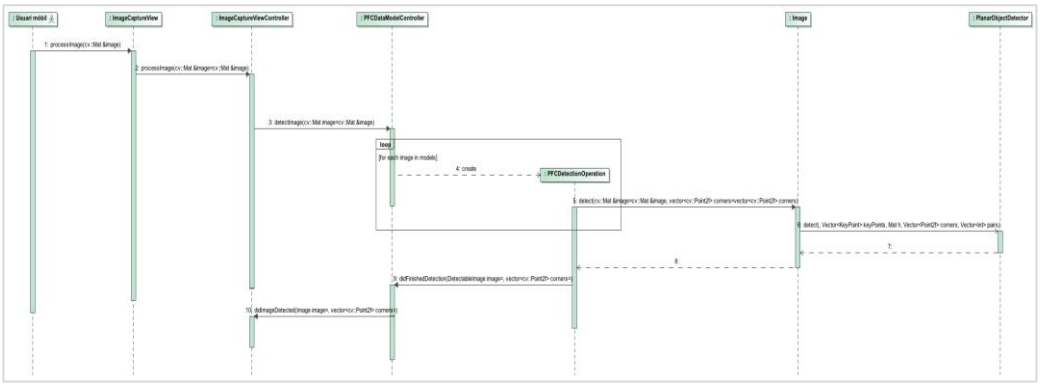

Figura 22. Diagrama de seqüència on es mostra el procés que es segueix per tal d'identificar una imatge. Pot observar-se com es genera una operació d'identificació per a cada imatge d'un model.

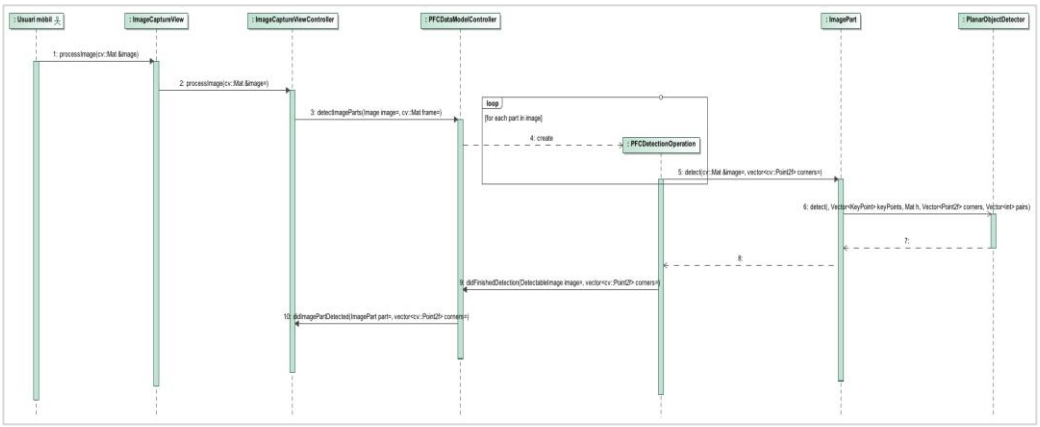

Figura 23. Diagrama de seqüència on es mostra el procés que es segueix per tal d'identificar la part d'una imatge ja detectada. Igual que per a la identificació d'una imatge, es genera una operació per intentar identificar cadascuna de les parts.

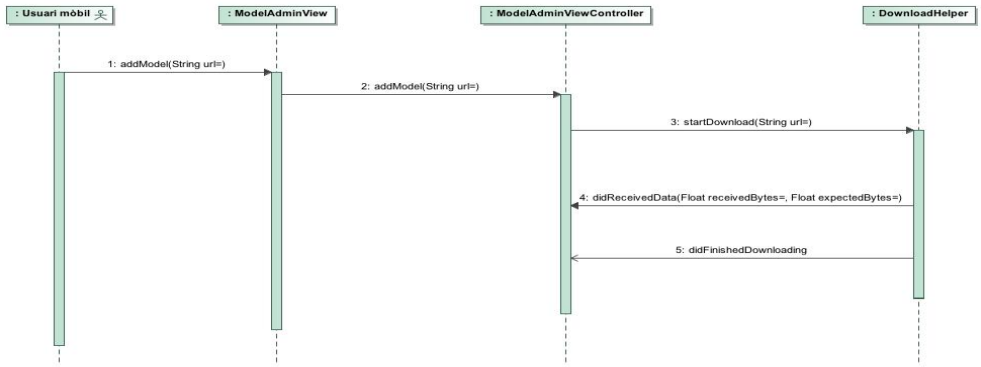

Figura 24. Diagrama de seqüència on es mostra el procés de descàrrega d'un model al dispositiu mòbil.

Un altre element important és la descàrrega de models d'Internet, a través d'una URL que apunti a un model mòbil creat per un usuari amb l'aplicació

d'entrenament. En aquest diagrama es pot observar l'ús de la classe DownloadHelper per a la descàrrega del model.

Detalls de la implementació

Donat el fet que el codi font de les aplicacions s'adjunta a l'entrega del projecte, en aquest capítol comentarem els detalls més significatius de la implementació de les aplicacions. També, en aquesta memòria, es pot veure a "Annex A. Mètode d'entrenament d'un detector" el codi font del mètode que empra les llibreries *OpenCV* per a entrenar un detector. De la mateixa manera, a "Annex B. Mètode de detecció", s'inclou el codi font del mètode que fa ús d'*OpenCV* per a detectar una imatge donat un *frame*.

Per a començar, parlarem de l'eina emprada per a desenvolupar tant l'aplicació d'entrenament com l'aplicació mòbil: *XCode 4.6*. Amés d'esser l'eina més adequada per a desenvolupar aplicacions natives en entorns *OSX* o *iOS*, proporciona moltes facilitats a l'hora de dissenyar les interfícies gràfiques de les aplicacions que es porten a terme, el qual és de vital importància per tal de poder centrar-nos el més aviat possible amb la lògica pròpia de les aplicacions.

Tal com ja s'ha explicat, per portar a terme l'entrenament i detecció s'han emprat les llibreries *OpenCV,* en la seva versió 2.4.4. Aquestes llibreries estan implementades en *C/C++*, però tot i això es poden fer servir amb *Objective C*, el llenguatge en que han estat programades les aplicacions d'aquest projecte, sense cap problema. És necessari configurar el projecte de forma adient, el qual implica el següent:

1. Descarregar i compilar les llibreries d'*OpenCV* amb l'arquitectura correcta que s'hagi d'emprar a la nostra aplicació. En el nostre cas, la nostra aplicació és compatible amb entorns *OSX* de 64 bits.
2. Configurar el compilador LLVM del projecte per tal que empri les llibreries de *C++ libc++*.
3. *Linkar* les llibreries estàtiques necessàries generades quan es compila *OpenCV*.

Finalment, cal tenir en compte què, per tal d'evitar errors de compilació que poden resultar bastant críptics, les capçaleres d'*OpenCV* que es requereixin s'han de definir al fitxer amb extensió *pch* del projecte *XCode*, tal com es mostra a la figura 25.

```
#import <Availability.h>

#ifndef __IPHONE_5_0
#warning "This project uses features only available in iOS SDK 5.0 and later."
#endif

#ifdef __cplusplus
#import <opencv2/opencv.hpp>
#import <opencv2/legacy/legacy.hpp>
#endif

#ifdef __OBJC__
    #import <UIKit/UIKit.h>
    #import <Foundation/Foundation.h>

#endif
```

Figura 25. Contingut del fitxer *pch* del projecte de l'aplicació mòbil.

Llibreries OpenCV

Pel que fa a les llibreries *OpenCV*, aquestes ens proporcionen les funcionalitats adients per tal d'aconseguir el nostre objectiu. Nosaltres emprem, tant a la fase d'entrenament com a la fase de detecció, els següents objectes propis de les llibreries *OpenCV*:

- *PlanarObjectDetector*. Aquest objecte és una classe *wrapper* dels objectes *LDetector* i *FernsClassifier*.
- *LDetector*. Es tracta de l'objecte capaç d'extreure els punts claus més estables d'una imatge, així com també els punts claus de la imatge en general.
- *FernsClassifier*. És el classificador que implementa l'algorisme *Random Ferns*. Es genera internament a la classe *PlanarObjectDetector* durant la fase d'entrenament, aplicant les transformacions necessàries sobre els punts clau més estables. Al final del procés, conté tota la informació que cal per tal d'identificar una imatge en concret.

Al model de l'aplicació mòbil hem pogut veure com cada objecte de tipus *DetectableImage* té associat un objecte de tipus *PlanarObjectDetector*. Emmagatzemar aquest objecte, com ja s'ha explicat, resulta ser un dels principals problemes als que s'han de fer front quan s'entrenen els models, donada la gran quantitat de memòria que pot arribar a consumir. La mida encara és més gran quan s'ha de guardar a un fitxer, ja que *OpenCV* solament ens permet fer-ho en format text, ja sigui *XML* o *YML*, el qual resulta un problema a l'hora de transferir el fitxer del model al dispositiu mòbil. Aquest fet ens ha fet optar per la compressió *gzip* per als fitxers dels models. Al tractar-se de fitxers de text, els ratis de compressió observats són bastant bons, oscil·lant entre 1:7 i 1:9. Com a exemple, el model emprat durant les proves, capaç d'identificar el quadre de

Las Meninas i a cinc dels personatges que hi apareixen, sense comprimir ocupa aproximadament 40MB, mentre que comprimit ocupa 7MB.

Extensions de classes

Amb *Objective C*, es poden portar a terme el que s'anomenen extensions de classes ja existents. La principal utilitat d'això, és la possibilitat de dotar d'una funcionalitat extra concreta, que s'emprarà de forma puntual, a una classe determinada. En altres casos, el més adient és recórrer als mecanismes d'herència si estem fent ús de llenguatges orientats a objecte.

Extensió de la classe NSImage

A l'aplicació d'entrenament, es porta a terme una extensió de la classe *NSImage* del *Cocoa Framework* (el *framework* sota el qual es desenvolupen les aplicacions natives amb interfície gràfica d'*OSX*). Aquesta extensió proporciona el següent:

- Un constructor, per tal de crear un nou objecte de tipus *NSImage* a partir d'un objecte de tipus *Mat*, propi d'*OpenCV*, i que representa una matriu.
- Un mètode estàtic que ens permet també crear un nou objecte de tipus *NSImage* a partir d'un objecte de tipus *Mat*.
- Una propietat que ens permet obtenir la representació de la imatge en un objecte tipus *Mat*, que és amb el tipus d'objecte amb el que treballen les llibreries d'*OpenCV* per tal de tractar imatges.
- Una propietat que ens permet obtenir la representació de la imatge en un objecte tipus *Mat*, igual que l'anterior, amb la diferència que aquesta correspon a la representació en blanc i negre de la imatge.

Amb aquesta funcionalitat, és fàcil treballar sempre amb objectes de tipus *NSImage*, per exemple quan afegim una nova imatge al model. Solament a l'hora de generar el model mòbil, requerim l'ús d'objectes de tipus *Mat* propis d'*OpenCV*, ja que són necessaris de cara a extreure els punts clau de la imatge.

Extensió de la classe NSData

Per tal de portar a terme la compressió dels models que ha de consumir l'aplicació mòbil, també s'ha portat a terme una extensió de la classe *NSData* d'*Objective C*. Aquesta classe representa una seqüència de bytes que, entre d'altres coses, es poden guardar a un fitxer, o pot ser generada a partir d'un fitxer. Quan acaba l'entrenament del model mòbil, aquest es guarda de forma temporal en un fitxer de text amb extensió *YML*, que són juntament amb els fitxers *XML*, el tipus de fitxer amb el que poden interactuar les llibreries *OpenCV*. El que es fa just després, és carregar aquest fitxer de text a un

objecte *NSData*. Posteriorment, emprant la funció *gzipDeflate*, que és la que hem implementat a la nostra extensió, obtenim un nou objecte de tipus *NSData* comprimit, que finalment guardem al disc en el que serà el fitxer final que representa el model, amb l'extensió *.md*. El fitxer *YML* generat anteriorment s'esborra del disc.

A l'aplicació mòbil hem emprat la mateixa extensió per tal de descomprimir un model descarregat. Just quan arranca l'aplicació, el primer que es porta a terme és la càrrega dels models existents a la memòria. Per tal de fer-ho, hem de carregar el fitxer model a un objecte *NSData*. Posteriorment, hem de descomprimir l'objecte emprant aquest cop el mètode *gzipInflate*, el qual ens retorna un altre objecte *NSData* descomprimit. Aquest objecte s'ha de guardar al dispositiu, ja que es tracta del fitxer *YML* que *OpenCV* és capaç de llegir i que ens servirà per a carregar el model. Un cop carregat el model, es borra el fitxer *YML* descomprimit per tal de que no ocupi espai innecessari al dispositiu.

Estructura del fitxer del model mòbil

Durant tota la memòria s'ha fet referència al model mòbil i, dintre d'aquest mateix capítol, s'ha comentat que realment és un fitxer *YML* que pot esser interpretat per les llibreries *OpenCV*. Tot i això, amb la funcionalitat que ens proporcionen aquestes llibreries, podem anar més enllà i guardar al model no solament objectes propis d'*OpenCV*, sinó també *Strings*, nombres o fins i tot els nostres propis objectes.

L'estructura interna que s'ha decidit per al model mòbil és la mateixa que s'ha presentat al model de l'aplicació mòbil. A continuació, s'especifica per nivell les dades que guardem al fitxer *YML* i de quin tipus són.

En el primer nivell del fitxer *YML* tenim la següent informació:

- Nom del model (*String)*
- Nombre d'imatges que conté el model *(Integer)*

Al segon nivell, ja dintre del model, tenim el llistat de quadres que conté el model, on cadascuna té la seva informació associada:

- Nom del quadre *(String)*
- Autor *(String)*
- Any *(Integer)*
- El detector de la imatge ja entrenat *(PlanarObjectDetector)*
- Nombre de parts que conté cada imatge *(Integer)*

Finalment, tenim un tercer nivell dintre de cadascuna de les imatges, el qual conté la informació associada a cadascuna de les parts:

- Nom de la part *(String)*
- Descripció *(String)*
- El detector de la imatge corresponent a la part ja entrenat *(PlanarObjectDetector)*

Paràmetres per a l'entrenament dels models

Per a portar a terme l'entrenament dels models, hi ha diversos paràmetres ajustables als mètodes que ens proporcionen els objectes d'*OpenCV* esmentats anteriorment. A continuació es presenta la llista de paràmetres més rellevants per a l'entrenament, i els valors que se'ls hi han assignat:

- Nombre de transformacions sobre la imatge original per a obtenir els punts clau més estables. Aquest paràmetre es defineix al constructor de l'objecte *LDetector* i el valor emprat durant la fase d'entrenament és de 10000.
- Nombre d'octaves. També és un paràmetre que forma part del constructor de l'objecte *LDetector*. El valor a la fase d'entrenament és 4. La piràmide de la imatge d'entrenament té #Octaves – 1 nivells.
- Nombre de punts estables a extreure. Amb els paràmetres anteriors, s'extreuen 200 punts estables a la fase d'entrenament.
- Mida del *patch*. El *patch* és la zona rectangular que envolta un punt clau. Per a l'entrenament, es generaran amb una mida de 16x16 píxels.
- Nombre de *Ferns* i mida dels *Ferns*. Per a cada punt clau, es generaran 30 *Ferns*, cadascun de mida 6. D'aquesta manera s'aconsegueixen generar models eficients en la detecció, i que no ocupen espais desproporcionats a memòria.
- Nombre de transformacions sobre cada *patch*. Aquest paràmetre s'ha establert amb un valor de 10000. D'aquesta manera aconseguirem un conjunt d'entrenament molt robust per a cadascuna de les classes (recordem que cada punt clau estable és una classe).

Scrapping sobre articles de la *Wikipèdia*

Per tal d'extreure informació de les parts d'una imatge determinada, s'ha emprat la tècnica de *scrapping* sobre l'article de la *Wikipèdia* que parla de la imatge en qüestió. Mitjançant aquesta tècnica, s'extreu de l'HTML de la plana corresponent a l'article la informació de cadascuna de les parts. Aquesta informació està bàsicament composta per la imatge corresponent a la part, i el peu de foto descriptiu de la imatge. L'estructura de

classes que s'ha seguit és la que es pot observar a la figura 5. Tenim una classe base anomenada *BasicScrapper*, la qual ens proporciona una sèrie de mètodes que les classes filles han d'implementar. D'aquesta manera, en un futur es podrien portar a terme altres implementacions diferents al *WikipediaScrapper* que, a partir d'una URL, retornin informació relativa a les parts d'una imatge.

L'elecció de portar a terme *scrapping* sobre la plana d'un article de la *Wikipèdia* ve motivada, principalment, perquè tot i que existeix una *API* per a consultar informació relativa a un article, cap de les cridades a aquesta *API* ens retorna la informació que necessitàvem en el nostre cas en una sola crida i de manera simple: les fotos d'un article, i el seu peu de foto associat.

Proves de detecció

Durant la implementació d'ambdues aplicacions, s'han portat a terme diverses proves per tal d'assegurar la qualitat de la detecció dels quadres del model així com de les seves parts. Hem portat a terme dos tipus de proves:

- Proves amb una aplicació de test desenvolupada per *OSX*, que ens permet donat un model mòbil, carregar la imatge que volem detectar. També ens permet ajustar diferents paràmetres de la detecció per comprovar-ne l'efectivitat.
- Proves amb el dispositiu mòbil. Aquestes són essencials, ja que d'una banda l'entorn *XCode* no permet portar a terme proves amb la càmera del dispositiu des del simulador. D'altra banda, és evident que l'aplicació ha de funcionar bé sobre el dispositiu, per la qual cosa aquestes proves són un requisit indispensable.

Proves amb l'aplicació de test

Com ja s'ha comentat, l'aplicació de test, la qual s'adjunta amb el codi font del projecte, permet provar l'efectivitat de l'algorisme de detecció, mentre que ajustem diversos paràmetres del procés per veure quins donen millors resultats.

Els paràmetres ajustables des de l'aplicació de test són els següents:

- *Detection treshold* (Umbral de detecció). Es tracta del percentatge de punts que s'han reconegut del model original, a partir del qual considerarem que la imatge ha estat detectada.
- *Keypoints* (punts clau). Es tracta del nombre de punts clau que s'hauran d'extreure de la imatge capturada.
- *Num. Octaves*. Aquest valor ens permet definir l'altura de la piràmide que es generarà, i que tindrà com a base la imatge capturada. A cada nivell de la piràmide, es reescala la imatge a 1:2. Quan es construeix el model, es treballa sobre una piràmide de 3 nivells que té com a base la imatge d'entrenament.
- *Scale factor*. La imatge capturada també es reescalarà, per tal d'intentar detectar-hi la imatge del model sobre la imatge reescalada.

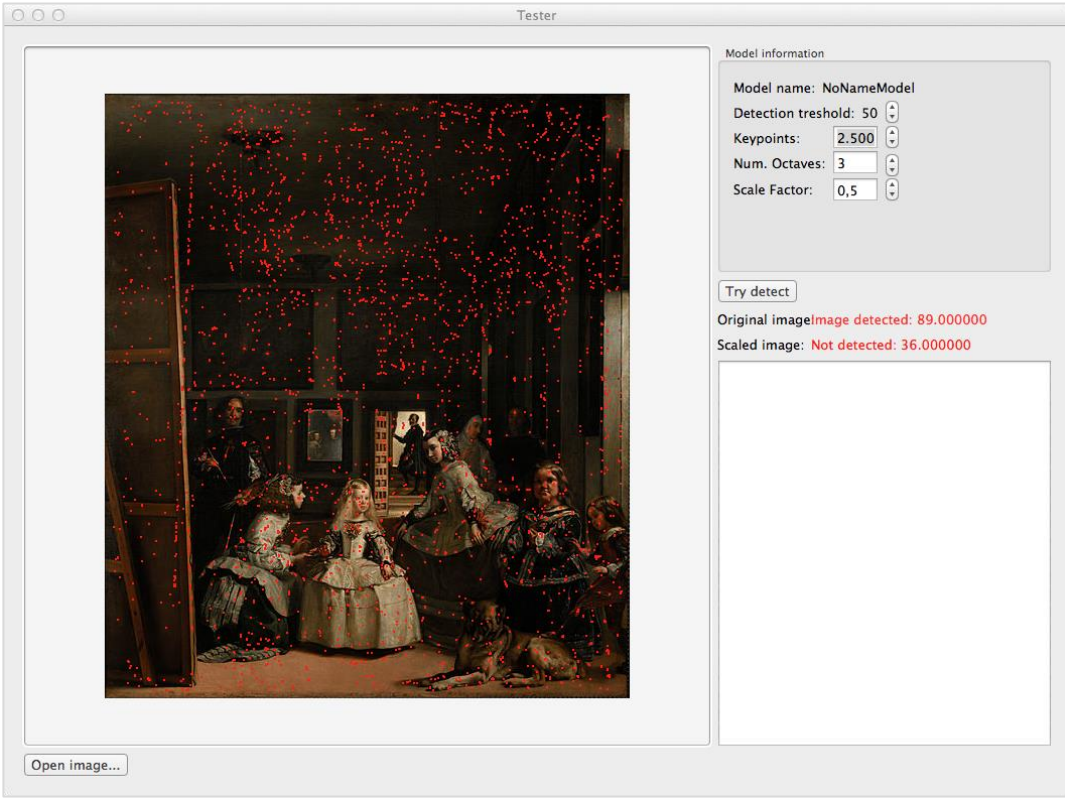

Figura 26. Pantalla de l'aplicació de test. Des del menú podem carregar un model mòbil, mentre que des de la part inferior podem carregar la imatge que ha de simular la captura des de la càmera del dispositiu. A la part dreta de la pantalla, veiem el nom del model carregat, i amés tenim la possibilitat d'editar els paràmetres de detecció. També apareixen els missatges que ens indiquen si la imatge ha estat detectada o no, tant a la versió original com a la versió escalada, i quines de les seves parts s'han detectat. Els punts vermells que es pinten sobre la imatge corresponen als punts clau extrets per tal d'indentar fer el *matching* amb els punts del model.

# Prova	# Punts clau	# Octaves	Escala	Percentatge detecció original	Percentatge detecció escalada	Percentatge detecció de la part	Percentatge detecció de la part escalada
1	2500	4	0,5x	91,5%	36,5%	31,5%	5,5%
2	1500	4	0,5x	91,5%	36,5%	31,5%	5,5%
3	1500	3	0,5x	90%	36,5%	31,5%	2,5%
4	1500	2	0,5x	82%	34,5%	31%	5,5%
5	1500	2	0,8x	82%	52%	29,5%	22,5%
6	1000	2	0,8x	82%	52%	2%	22,5%
7	800	4	0,8x	82,5%	58,5%	32,5%	23%
8	800	4	1,5x	82,5%	29,5%	32,5%	17,5%
9	1500	4	1,5x	91,5%	14,5%	31,5%	5,5%
10	2500	4	1,5x	91,5%	3,5%	31,5%	4%

Figura 27. Taula amb els resultats de les diferents proves realitzades amb l'aplicació de test.

A la figura 27, es pot veure la taula amb les diferents proves realitzades amb un model capaç de reconèixer el quadre de *Las Meninas* així com al personatge de la

Infanta Margarita. Els percentatges de detecció representen el nombre de punts del model entrenat que s'han trobat a la imatge capturada.

Vistes aquestes proves, podem dir que l'escala de la imatge capturada sembla un factor bastant important respecte a l'efectivitat de la detecció. Quan l'escala de la imatge és inferior a l'original, la detecció és menys efectiva, especialment la de les parts, ja que els percentatges de detecció de la imatge global quasi no varien, exceptuant els casos en que el nombre d'octaves (que determinen la mida de la piràmide que pren com a base la imatge original) es menor que l'original (4). En canvi, amb escales superiors, l'efectivitat de la detecció es veu reduïda considerablement. Tot i això, al dispositiu mòbil capturem una imatge de 720x1280, per tant entrenant models a partir d'imatges d'aquesta mida o fins i tot una mica inferiors, ben segur que dóna bons resultats si s'enfoca amb la càmera a tot el quadre.

Baixar massa el nombre de punts clau extrets de la imatge capturada sembla que pot conduir a una mica d'inestabilitat de l'algoritme com es pot veure a la prova 6. Amés també es redueix l'eficàcia de la detecció, especialment pel fet que d'aquesta manera és menys probable que els punts extrets es corresponguin amb els de la imatge emprada a l'entrenament.

Proves amb l'aplicació mòbil

Amb les proves efectuades sobre l'aplicació mòbil, podem veure el comportament de la detecció en l'entorn real d'execució. A l'aplicació mòbil, després de les proves realitzades a l'apartat anterior i de diverses proves realitzades amb el dispositiu, s'ha establert que com a mínim un 20% de punts han de coincidir respecte als de la imatge original per tal de considerar-la com a detectada. Aquest valor ha funcionat bé durant totes les execucions portades a terme, ja que així aconseguim detectar la zona exacta on s'ha trobat la imatge sobre la imatge capturada, i donem un cert marge a possibles variacions durant la captura (el moviment del dispositiu o la quantitat de llum de l'entorn). Prova de l'efectivitat d'aquest llindar, és l'exactitud amb la que es col·loquen els *pins* de color verd quan es detecten les parts de les imatges.

Amb l'aplicació mòbil les proves són més tedioses de realitzar que no pas amb l'aplicació de test, però són més realistes. Emprant el mateix model que a l'aplicació de test, a la figura 28 es pot apreciar el *log* de l'aplicació des de que inicia la càmera fins que es detecta el quadre de *Las Meninas* així com la imatge de *La Infanta Margarita*.

2013-05-18 16:02:31.653 Art Detector[32174:907] camera available: YES
2013-05-18 16:02:31.694 Art Detector[32174:907] [Camera] device connected? YES
2013-05-18 16:02:31.695 Art Detector[32174:907] [Camera] device position back
2013-05-18 16:02:31.703 Art Detector[32174:907] layout preview layer
2013-05-18 16:02:31.704 Art Detector[32174:907] flip bounds
2013-05-18 16:02:31.706 Art Detector[32174:907] [Camera] created AVCaptureVideoDataOutput at 20 FPS
2013-05-18 16:02:32.462 Art Detector[32174:907] deviceOrientationDidChange: 1
2013-05-18 16:02:32.464 Art Detector[32174:907] rotate..
2013-05-18 16:02:32.465 Art Detector[32174:907] layout preview layer
2013-05-18 16:02:34.192 Art Detector[32174:1903] Name: Las Meninas; Coincidence: 0.315000; Detected: YES
2013-05-18 16:02:36.134 Art Detector[32174:1903] Name: Infanta Margarita; Coincidence: 0.080000; Detected: NO
2013-05-18 16:02:38.051 Art Detector[32174:1903] Name: Infanta Margarita; Coincidence: 0.235000; Detected: YES
2013-05-18 16:02:40.142 Art Detector[32174:1903] Name: Infanta Margarita; Coincidence: 0.240000; Detected: YES
2013-05-18 16:02:42.621 Art Detector[32174:1103] Name: Infanta Margarita; Coincidence: 0.215000; Detected: YES
2013-05-18 16:02:45.656 Art Detector[32174:5403] User moved the device. Y value is 0.061626. X value is -0.027614

Figura 28. Podem apreciar com, des de que s'inicia la captura de *frames* amb la càmera fins a que es detecta el quadre de *Las Meninas* passen uns 3 segons. Un cop es detecta el quadre, la detecció de *La Infanta Margarita* sobre la imatge capturada triga uns 2 segons.

El resultat, visible des de la pantalla de l'aplicació mòbil, és el que es mostra a la figura 29. La prova ha estat realitzada enfocant amb la càmera al quadre de *Las Meninas*, mostrat a la pantalla de l'ordinador. El fet que la imatge aparegui moguda és degut al moviment del dispositiu durant la captura de pantalla.

Figura 29. Resultat visible des de la vista de la captura del dispositiu mòbil un cop identificades les dues imatges.

Falta veure un detall important, i és observar com funciona la detecció si la imatge es captura des d'un angle que no sigui frontal. Aquesta prova no podíem fer-la amb l'aplicació de test, però els resultats amb l'aplicació mòbil es poden apreciar a les figures 30 i 31. Es pot veure com el temps de detecció de *La Infanta Margarita* augmenta respecte a si es fa una captura frontal. El temps de detecció del quadre segueix essent molt ràpid. Des de la captura del primer *frame*, es triga 2 segons en detectar el quadre.

2013-05-18 16:18:02.220 Art Detector[32174:5403] Name: Las Meninas; Coincidence: 0.195000; Detected: NO
2013-05-18 16:18:04.277 Art Detector[32174:606b] Name: Las Meninas; Coincidence: 0.200000; Detected: YES
2013-05-18 16:18:06.947 Art Detector[32174:606b] Name: Infanta Margarita; Coincidence: 0.145000; Detected: NO
2013-05-18 16:18:08.994 Art Detector[32174:606b] Name: Infanta Margarita; Coincidence: 0.045000; Detected: NO
2013-05-18 16:18:11.686 Art Detector[32174:606b] Name: Infanta Margarita; Coincidence: 0.170000; Detected: NO
2013-05-18 16:18:14.370 Art Detector[32174:606b] Name: Infanta Margarita; Coincidence: 0.020000; Detected: NO
2013-05-18 16:18:16.464 Art Detector[32174:606b] Name: Infanta Margarita; Coincidence: 0.050000; Detected: NO
2013-05-18 16:18:18.537 Art Detector[32174:606b] Name: Infanta Margarita; Coincidence: 0.160000; Detected: NO
2013-05-18 16:18:21.089 Art Detector[32174:606b] Name: Infanta Margarita; Coincidence: 0.135000; Detected: NO
2013-05-18 16:18:23.392 Art Detector[32174:5fdf] Name: Infanta Margarita; Coincidence: 0.080000; Detected: NO
2013-05-18 16:18:25.178 Art Detector[32174:5fdf] Name: Infanta Margarita; Coincidence: 0.030000; Detected: NO
2013-05-18 16:18:27.368 Art Detector[32174:5fdf] Name: Infanta Margarita; Coincidence: 0.200000; Detected: YES
2013-05-18 16:18:29.665 Art Detector[32174:5fdf] Name: Infanta Margarita; Coincidence: 0.150000; Detected: NO

2013-05-18 16:18:31.941 Art Detector[32174:606b] Name: Infanta Margarita; Coincidence: 0.175000; Detected: NO

Figura 30. Resultats de la detecció amb el dispositiu mòbil, capturant la imatge des d'un angle diferent respecte la prova anterior.

Figura 31. Resultat de la detecció amb el dispositiu des d'un angle no frontal

Per últim, falta veure quin és el resultat de la detecció en cas de fer-la des d'un punt de vista més llunyà respecte al quadre, de forma que aquest no ompli tot el *frame*. A les figures 32 i 33 es mostren els temps de detecció així com la captura de pantalla del dispositiu.

2013-05-19 16:11:55.493 Art Detector[32837:907] camera available: YES
2013-05-19 16:11:55.617 Art Detector[32837:907] [Camera] device connected? YES
2013-05-19 16:11:55.619 Art Detector[32837:907] [Camera] device position back
2013-05-19 16:11:55.635 Art Detector[32837:907] layout preview layer
2013-05-19 16:11:55.637 Art Detector[32837:907] flip bounds
2013-05-19 16:11:55.640 Art Detector[32837:907] [Camera] created AVCaptureVideoDataOutput at 20 FPS
2013-05-19 16:11:56.684 Art Detector[32837:907] deviceOrientationDidChange: 1
2013-05-19 16:11:56.686 Art Detector[32837:907] rotate..
2013-05-19 16:11:56.687 Art Detector[32837:907] layout preview layer
2013-05-19 16:11:56.866 Art Detector[32837:907] Received memory warning.
2013-05-19 16:12:00.455 Art Detector[32837:660b] Name: Las Meninas; Coincidence: 0.480000; Detected: YES
2013-05-19 16:12:02.312 Art Detector[32837:1103] Name: Infanta Margarita; Coincidence: 0.235000; Detected: YES
2013-05-19 16:12:04.125 Art Detector[32837:1803] Name: Infanta Margarita; Coincidence: 0.315000; Detected: YES

2013-05-19 16:12:05.957 Art Detector[32837:1803] Name: Infanta Margarita; Coincidence: 0.310000; Detected: YES

Figura 32. Resultats de la detecció amb el dispositiu mòbil més allunyat de la imatge original. Podem veure que des del moment en que s'inicia la captura amb la càmera, la detecció tant del quadre de *Las Meninas* com del personatge de *La Infanta Margarita* és bastant ràpida.

Figura 33. Resultat de la detecció amb el dispositiu mòbil, enfocant el quadre a major distància que a la primera prova.

Conclusions

Sens dubte, el més complicat del projecte ha estat la seva implementació. Primer de tot, degut a que era la meva primera experiència en el desenvolupament d'aplicacions mòbils. Tot i conèixer els patrons de disseny emprats, és cert que s'han de tenir en compte altres particularitats d'aquests dispositius: no es tracta d'ordinadors com els que tenim a casa, que són molt més potents. Aquí hem de "jugar" al màxim amb l'eficiència del codi generat per tal de no desaprofitar el potencial que ens proporciona el dispositiu. Per exemple, les llibreries *OpenCV* ens donen la possibilitat de capturar vídeo a diferents resolucions. En cas d'escollir la més alta, el processament de cada *frame* resultava amb un *Memory Warning* en temps d'execució, el qual provocava que l'aplicació fos molt lenta. El problema va desaparèixer al reduir la resolució del vídeo a 720x1280.

L'altre punt clau en quan a la dificultat de la implementació, ha estat l'ús de les llibreries *OpenCV*. És cert que aquestes ens abstreuen de tasques complexes com la implementació de l'algorisme *Random Ferns*, però tot i això, és convenient tenir un bon coneixement de com funciona l'algorisme, per tal d'identificar possibles errades en la detecció. Exemple d'això, és el fet que la imatge d'entrenament hauria de tenir una mida similar a la de la imatge capturada per tal d'extreure els punts claus més adients respecte als punts clau més estables extrets durant l'entrenament. Aquesta conclusió parteix dels problemes que es van trobar a l'hora de la detecció de les parts. Inicialment s'emprava la imatge obtinguda de la *Wikipèdia* per tal d'entrenar el detector capaç de reconèixer una part. Els resultats eren bastant negatius, ja que s'havia d'acostar molt el dispositiu mòbil al quadre per tal de detectar les parts, el qual a un museu podria ser una situació complicada, degut a que el més normal és estar una mica lluny del quadre per visualitzar-lo bé. Emprant l'aplicació de test, es van pintar els punts extrets de la imatge capturada i es va observar que a les zones de les parts que es volien identificar, eren molt diferents dels punts estables extrets de la imatge de la *Wikipèdia* emprats per a l'entrenament. Es va provar d'entrenar el detector de la part amb la imatge corresponent retallada des de la imatge emprada per entrenar el quadre complet, i els resultats van esser totalment diferents. El rati de detecció va augmentar molt respecte a l'observat fins aquell moment, i el comportament ja era semblant al que s'observava a l'hora de detectar el quadre complet: es podia detectar la part des d'una distància més gran i fins i tot des d'un angle diferent al frontal.

Això no vol dir que el que s'ha portat a terme en quan a extreure les parts del quadre de la *Wikipèdia* sigui inútil. El cert és que la descripció de les parts que s'obté encara ens serveix, i pot existir alguna imatge que també ens serveixi per tal d'entrenar el detector d'una de les parts del quadre. Amés, el disseny de l'aplicació d'entrenament

ens permet implementar altres classes que heretin de la classe *BasicScrapper* i que tinguin la capacitat d'extreure informació d'una altra font *on-line*, ja sigui perquè ens proporciona millors imatges o perquè ens proporciona una informació més fiable.

Finalment, cal dir que la idea presentada presenta certs aspectes millorables, però degut a que el temps per a realitzar el projecte és limitat, es tractaria de millores a mitjà o llarg termini. Una de les principals carències és el format en que *OpenCV* ens permet guardar els objectes, que com ja s'ha comentat és *XML* o *YML*, ocupant un espai notable en disc en cas que es vulgui desar un model capaç d'identificar varis quadres. Trobar la manera de desar aquests objectes en format binari podria reduir força la mida del model generat, i encara milloraria més en cas d'aplicar-hi compressió.

Un altre camí a triar podria ser la identificació *on-line* de les imatges, tal com el que porta a terme l'aplicació *Shazam* per a identificar una cançó. Això provocaria un cert canvi en el disseny de l'aplicació mòbil, ja que seria desorbitat enviar cada *frame* a un servei web remot. L'ideal en aquest cas seria disposar d'un botó que, al polsar-lo, pugui enviar un nombre limitat de *frames* capturats a un *WebService* remot, i que aquest respongui amb el resultat de la identificació. El principal problema d'aquesta solució seria el consum de tràfic de xarxa de l'usuari, especialment en cas d'estar connectat amb *3G*, però emprant compressió per tal de transferir les dades dels *frames*, podríem minimitzar l'ample de banda que es consumeix.

Annex A. Mètode d'entrenament d'un detector

```
- (cv::PlanarObjectDetector) trainDetectorForImage: (NSImage *) img
{

    cv::PlanarObjectDetector detector;

    // Convert image to grayscale
    cv::Mat grayScale;

    cv::cvtColor([img CVMat], grayScale, CV_BGR2GRAY);

    NSSize size = [img size];

    // Creating LDetector
    // - 4 octaves
    // - 10000 transformations
    // - 16x16 patch size, unless image it's smaller than 16x16
    cv::Size patchSize(MIN(PATCH_SIZE, (int)(size.width)/4), MIN(PATCH_SIZE, (int)(size.height)/4));
    cv::LDetector ldetector(7,20,NUM_OCTAVES,STABLE_KEYPOINT_TRANSFORMATIONS, patchSize.width,
2);

    // Bluring image for better performance against mobile camera noise
    vector<cv::Mat> imgpyr;
    int blurKSize = 3;
    double sigma = 0;

    cv::GaussianBlur(grayScale, grayScale, cv::Size(blurKSize, blurKSize), sigma, sigma);

    // Build image pyramid
    cv::buildPyramid(grayScale, imgpyr, ldetector.nOctaves-1);

    // Getting 200 most stable keypoints
    vector<cv::KeyPoint> imgKeypoints;
    cv::PatchGenerator gen(0,256,2,true,0.8,1.2,-CV_PI/2,CV_PI/2,-CV_PI/2,CV_PI/2);

    ldetector.getMostStable2D(grayScale, imgKeypoints, STABLE_KEYPOINTS, gen);

    // Training Ferns classifier
    // - 30 ferns of size 6
    // - Generate 10000 samples for training each class
    detector.train(imgpyr, imgKeypoints, patchSize.width, NUM_FERNS, FERNS_SIZE, TRAINING_SET_SIZE,
ldetector, gen);

    return detector;

        }
```

Figura 34. Implementació del mètode d'entrenament del detector d'una imatge.

Annex B. Mètode de detecció

```
- (BOOL) detect:(cv::Mat &)sourceImage corners:(std::vector<cv::Point2f> &) dstCorners {

    detector.setVerbose(false);

    cv::Mat image;
    cv::Mat H;
    vector<int> pairs;
    vector<cv::KeyPoint> imgKeypoints;

    // Create the LDetector. We will only use the octaves parameter
    // for determining the image pyramid size
    cv::LDetector ldetector(7,20,4,10000, 16, 4);

    vector<cv::Mat> imgpyr;

    int blurKSize = 3;
    double sigma = 0;

    // Blur the captured image and build the image pyramid
    cv::GaussianBlur(sourceImage, sourceImage, cv::Size(blurKSize, blurKSize), sigma, sigma);
    buildPyramid(sourceImage, imgpyr, ldetector.nOctaves-1);

    // Getting 2000 keypoints from the source image
    ldetector(imgpyr, imgKeypoints, 2000, true);

    // Get if we have any coincidence between the captured image extracted
    // keypoints and the trained detector.
    bool found = detector(imgpyr, imgKeypoints, H, dstCorners, &pairs);

    // If we have found keypoints pairs coincidences, we must check for the %
    // of keypoints detected in our image
    if(found){
        int coincidences = (int)pairs.size() / 2;
        int modelKeypoints = (int)detector.getModelPoints().size();

        float coincidencePercentage = (float)coincidences / (float)modelKeypoints;

        NSLog(@"Name: %@; Coincidence: %f; Detected: %s", self.name, coincidencePercentage,
(coincidencePercentage >= MIN_COINCIDENCE) ? "YES" : "NO");

        return coincidencePercentage >= MIN_COINCIDENCE;
    }

    return false;

            }
```

Figura 35. Implementació del mètode de detecció al dispositiu mòbil.

Bibliografia

Apple Inc. (2013). *iOS Developer Library*. Obtingut de https://developer.apple.com/library/ios/navigation/

Díaz Mas, L. (2011). *OpenCV 2.X: Reading/Writing from/in markup languages (XML, YAML)*. Obtingut de http://plagatux.es/2011/08/opencv-2-x-readingwriting-infrom-markup-languages-xml-yaml/

Las Meninas. (2013). Obtingut de http://es.wikipedia.org/wiki/Las_Meninas

NSDataCategory. (s.f.). Obtingut de CocoaDev: http://cocoadev.com/wiki/NSDataCategory

Mark, D., Nutting, J., LaMarche, J., & Olsson, F. (2012). Beginning iOS 6 Development: Exploring the iOS SDK.

Özuysal, M., Calonder, M., Lepetit, V., & Fua, P. (2010). *Fast Keypoint Recognition using Random Ferns*. Obtingut de Ecole Polytechnique Fédérale de Laussane: http://cvlabwww.epfl.ch/~lepetit/papers/ozuysal_pami10.pdf

Open CV. (2013). *OpenCV Documentation*. Obtingut de http://opencv.org/documentation.html

Sadun, E. (2012). The iOS 5 Developer's Cookbook: Core Concepts and Essential Recipes for iOS Programmers. (3rd edition).