Grau d'Enginyeria Informàtica

Aprenentatge Automàtic

Landscape Image Classification

Joan Balaguer Llagostera
Jaume Adrover Fernández

19 de novembre de 2023

Índex

1 Introducció

En aquesta pràctica el nostre objectiu és crear un model SVM tal que donada una imatge, pugui classificar-la correctament en una de les següents classes:

- **1. Bedroom**
- **2. Coast**
- **3. Forest**
- **4. Highway**
- **5. Industrial**
- **6. Inside City**
- **7. Kitchen**
- **8. Living Room**
- **9. Mountain**
- **10. Office**
- **11. Open Country**
- **12. Store**
- **13. Street**
- **14. Suburb**

Aquestes imatges són reals i quasi no han estat pretractades. Per tant, haurem d'analitzar i extreure característiques de cada imatge del dataset per tal que aquestes siguin utilitzables per a la creació del nostre model SVM. Per tant, el primer que haurem de fer és analitzar a fons el dataset d'imatges.

1.1 Dataset

El primer que vàrem detectar quan analitzàvem les imatges és que no totes tenien la mateixa mida. Això provoca que algunes tinguin més pixes que altres i, per tant, sigui un dataset inutilitzable d'aquesta forma. Per tant, el primer que farem serà fer un *resize* a totes, analitzant quina és la imatge més petita de tot el dataset per després retallar la resta a aquesta mida. La imatge més petita és de 200x200 px i, per tant, una vegada fet el *resize* totes tenen aquesta mida.

Una vegada fet el *resize* de les imatges, veiem que el nostre dataset està separat en *train* i *test*, on cada carpeta té una subcarpeta amb cada classe d'imatge. Cal puntualitzar que dintre de la carpeta de test existeix una carpeta de la classe *livingRoom* les quals estan anotades com a conflictives. Revisant les imatges i comparant-les amb les de la carpeta *livingRoom* hem vist que simplement són un subconjunt de les imatges que es troben en aquesta carpeta. Per aquest motiu, hem suprimit la carpeta, ja que són dades duplicades. El tractament previ a l'entrenament dels models que fem del dataset, es pot trobar al notebook **data_prepare.ipynb**. Una vegada feta aquesta consideració, anem a veure quantes imatges tenim per a cada classe:

	Train	***Test***
Bedroom	100	116
Coast	100	260
Forest	100	228
Highway	100	160
Industrial	100	211
Inside City	100	208
Kitchen	100	110
Living Room	100	189
Mountain	100	274
Office	100	115
Open Country	100	310
Store	100	215
Street	100	192
Suburb	100	141
Tall Building	100	256

Com podem apreciar, hi ha el mateix nombre d'imatges per a entrenar cada classe, de manera que no hi haurà desigualtats a l'hora de classificar. Al test sí que hi ha un nombre diferent d'imatges per a cada classe, fet que no ens afecta en principi, ja que el model no serà entrenat amb aquestes dades. Pel que fa al format de les imatges, podem veure que totes segueixen un Gray Scale. Vegem algun exemple:

Figura 1: Tres fotos d'exemple del dataset

1.2 Model

El nostre model escollit per a la resolució d'aquest problema és un SVM, millor conegut com a *Support Vector Machines*. Aquest vendrà de la llibreria Sklearn.

2 Preparació de dades

En aquesta secció, ens encarregarem de preparar totes les dades amb l'objectiu de poder passar-les directament al model. El primer que hem de fer és veure si totes les imatges tenen el mateix tamany, ja que l'input podria ser un factor de decisió del model. No volem que el tamany influeixi així que farem un *resize* a totes del mínim tamany que trobem.

Hem creat un script per a poder veure la dimensió més petita que es pugui apreciar, que en el nostre cas es 200(aquesta part està millor explicada al *Jupyter Notebook*). Per tant, farem un resize a totes les imatges per a que tenguin un tamany de 200x200.

Posteriorment, pegarem una ullada al dataset en general i veurem si podem extreure alguna conclusió. D'aquest pas, ens hem fixat en que la carpeta **living Room (Case Conflict)** pareix la mateixa que **livingRoom**. Però a simple vista no podem afirmar-ho ja que es podrien haver vist modificacions en la matriu, per molt petites que fossin. Per això, hem creat un script que fa el següent:

Algorithm 1 Image Comparison Algorithm

```
Output: None

for each filename in os.listdir(test_living_room2) do
    for each filename2 in os.listdir(test_living_room) do
        if filename == filename2 then
            image1 = readImage(test_living_room + '/' + str(filename))
            image2 = readImage(test_living_room2 + '/' + str(filename2))

            if size(image1)==size(image2) then
                difference = subtractImages(image1, image2)

                if countNonZero(difference)==0 then
                    print("The images are identical.")
                else
                    print("The images are not identical.")
                end if
            else
                print("The images have different shapes.")
            end if
        end if
    end for
end for
```

Resumint, l'algoritme fa el següent:

- **1. Mira si a la carpeta livingRoom hi ha la imatge actual dels casos conflictius(comparant el nom)**
- **2. Si han coincidit en el cas anterior, llegeix les imatges i mira si els tamanys són iguals**
- **3. Si els tamanys són iguals, resta les dues imatges en forma de matriu i emmagatzema el resultat.**
- **4. Finalment, si trobes algun nombre que sigui diferent de 0, significa que un dels píxels de les imatges era diferent i, per tant, les imatges no eren idèntiques.Imprimeix: 'The images are identical'**

Finalment, quan executam l'algoritme veim que totes les imatges són idèntiques i, per tant, podem eliminar la carpeta corresponent amb la instrucció *shutil.rmtree()*. Després d'haver acabat aquesta operació, totes les imatges estan preparades per a la seva introducció dins el model.

3 Workflow

En aquest pas, definirem com funciona tota la nostra estructura de dades, il·lustrant amb un procés visual, a més d'una detallada descripció. Constam de les següents tres principals parts:

- **1. Feature Extraction**: en aquesta part, hem definit una instància de la classe **FeatureTransformer** que s'encarrega d'aplicar totes les transformacions i d'emmagatzemar-les dins el vector de característiques.
- **2. Model Training**: l'entrenament consta del **GridSearch** i un entrenament ja més específic amb els millors paràmetres trobats.
- **3. Model Evaluation**: finalment, passam el conjunt de **x_test** i obtenim un % d'accuracy.

3.1 Feature Extraction

En aquesta etapa de creació del nostre model, ens encarregam de passar totes les imatges a un vector de característiques. Vegeu el fluxe que seguim:

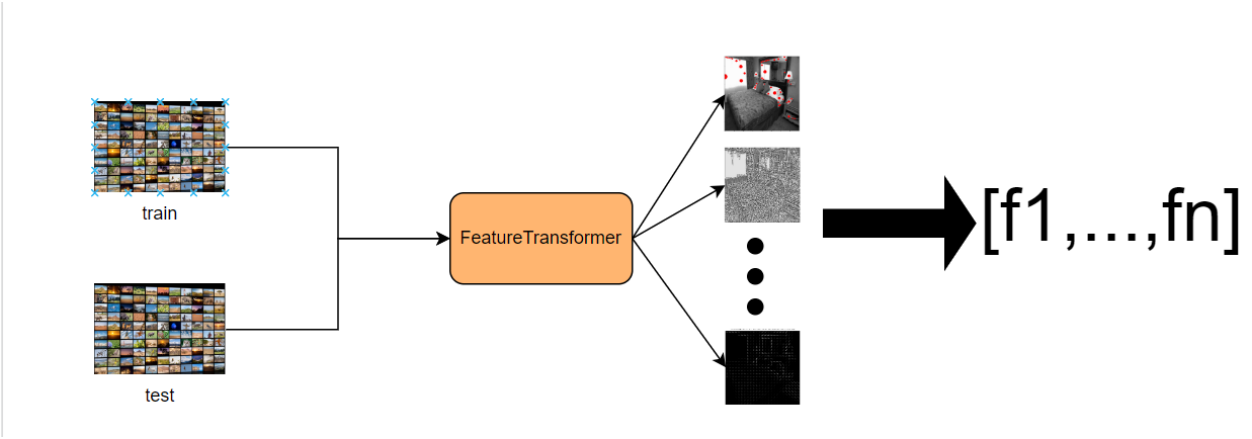

Figura 2: Procés d'extracció de Features

Ens encarregam de fer les següents passes:

- **1. Seleccionar una imatge**
- **2. Dividir la imatge per 255(tenir valors entre 0,1)**
- **3. Aplicar la transformació/ns corresponents i emmagatzemar resultat**
- **4. Afegir resultat al vector que correspon(X_train o X_test)**
- **5. Emmagatzemar el nombre de la classe com a etiqueta(Y_train o Y_test)**
- **6. Si queden imatges torna a la passa 1**

3.2 Model Training

El model training s'encarrega de entrenar el model amb el vector de característiques obtinguts a l'etapa anterior. A més, passam diferentes combinacions de paràmetres per a veure quina és la millor combinació. Vegeu la següent imatge:

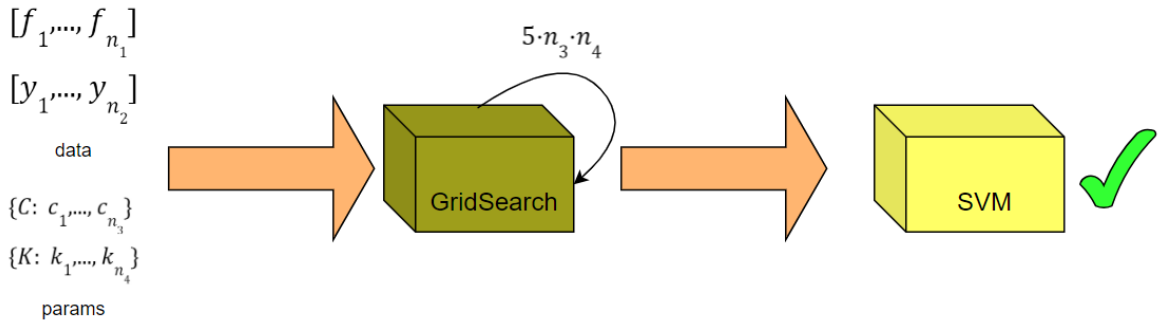

Figura 3: Procés d'obtenció d'un bon model amb paràmetres corresponents

L'algoritme és el següent:

- **1. Tria una combinació dels paràmetres**
- **2. Entrena el model amb aquella configuració corresponent, passant-li x_train,y_train. Això es fa 5 pics amb diferents folds(tamany de particions del dataset)**
- **3. Si queden combinacions restants, retorna a la passa 1**
- **4. Acabat tot el procés d'aprenentatge amb diferents paràmetres, imprimeix el millor estimador(combinació de paràmetres).**
- **5. Construeix el model amb els millors paràmetres trobats.**

3.3 Model Evaluation

En aquesta darrera i no menys interessant fase, ens encarregarem d'evaluar el model que hem obtingut a base de cercar la millor combinació de paràmetres amb el **GridSearch** de *Sklearn*. Bàsicament, li passarem la informació de test i intentarem que predigui els resultats correctes

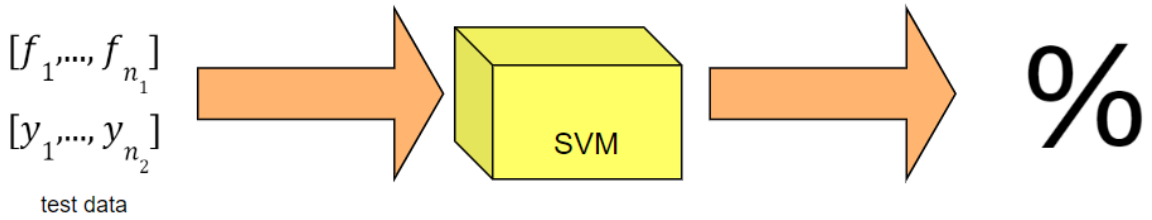

Figura 4: Procés d'evaluació final del model

L'algoritme és el següent:

- **1. Intentar prediure els resultats de tests i guardar-los dins una variable *y_meu***
- **2. Utilitzar la funció *accuracy_score* de Sklearn passant-li *y_test,y_meu***
- **3. Finalment, imprimir el resultat** $accuracy \in [0,1]$ **i mostrar la matriu de confusió**

El resultat imprès és un nombre comprès en aquest interval $[0,1]$, on 0 representa un 0% i 1 representa una predicció perfecta, 100%. També obtenim una matriu de confusió com la següent:

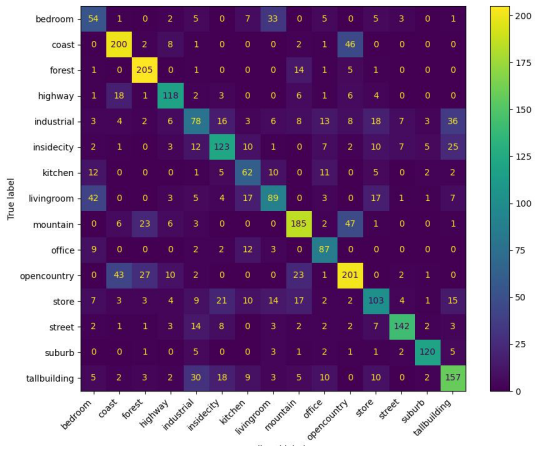

Figura 5: Matriu de confusió per al model HOG

4 Fase d'experimentació

En aquesta secció, ens encarregarem d'escollir quines features utilitzarem en el nostre model. Això ho hem fet mitjançant el desenvolupament de diferents Jupyter Notebooks dins la carpeta **testing**. Dins aquesta carpeta hi ha totes les combinacions possibles.

4.1 Feature visualization

En aquesta fase, ens encarregarem d'aplicar diferentes transformacions sobre una imatge i fer un display del resultat. Aplicarem les següents transformacions:

- **1. HOG(Histogram Of Oriented Gradient)**
- **2. Canny Edge**
- **3. LBP(Local Binary Pattern)**
- **4. Fast Corner**
- **5. Harris Corner**
- **6. Hessian Matrix**

La imatge original és la següent:

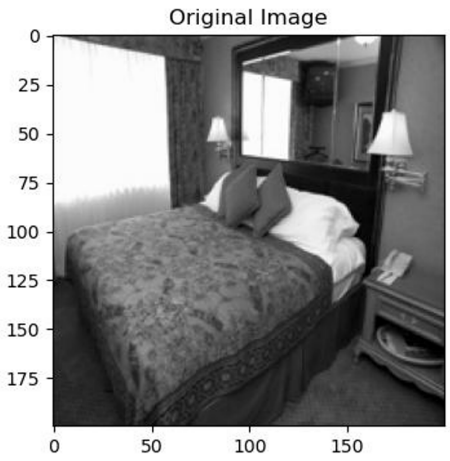

Figura 6: Figura Original

Com podem veure a simple vista, les que pareixeran més útils seran les de **Canny Edge** i **LBP**. Tot i això, només hem provat amb una imatge, per el que no tenim la certesa.

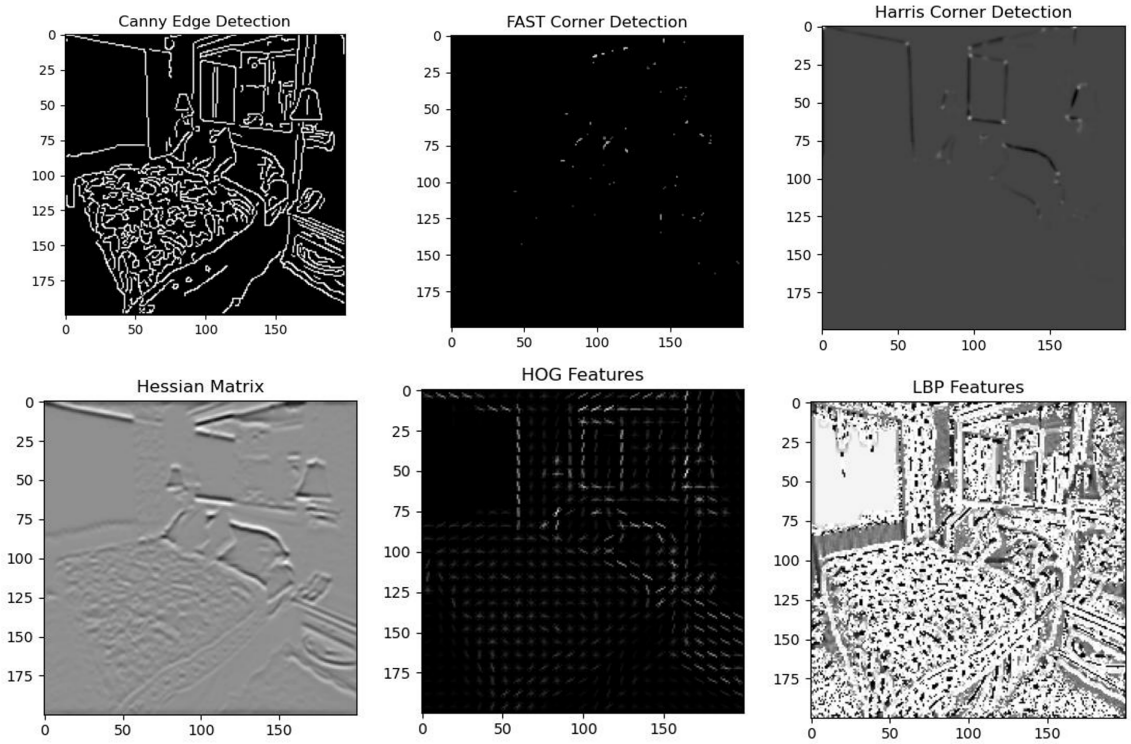

Figura 7: Diferents transformacions aplicades a la imatge de la figura 5

4.2 Feature testing

En aquesta fase, el nostre objectiu és avaluar i extreure les millors característiques. Cercam el millor % amb les millors configuracions del Support Vector Machines. Aquí la llista dels candidats a competir i els seus corresponents resultats:

Feature	***Best params***	***Accuracy***
HOG	C:1, kernel: poly	64.3%
LBP	C:10,kernel:'rbf'	32.7%
Canny Edge	C:10,kernel:'rbf'	35%
Harris Corner	C:100,kernel:'rbf'	32.3%
Hessian Matrix Determinant	C:10,kernel:'sigmoid'	33.1%
Corner Fast	C:10,kernel:'rbf'	18.9%
Multiblock LBP	C:100,kernel: 'rbf'	5.72%

Fetes totes les demostracions individuals, escollirem les dues millors i les combinarem per a obtenir un feature vector amb el següent format:

$$f_0(x) = HOG$$

$$f_1(x) = Canny$$

$$\vec{x} = [x_0, x_1 \mid x_0 = f_0(x') \wedge x_1 = f_1(x')]$$

En resum, concatenam el resultat de les tranformacions en l'ordre [HOG,Canny]. Aquesta sirà l'entrada que correspondrà a la variable **x_train** i la que es menjarà el model. Anem a veure la precisió final del model que combina les dues features anteriors:

- **1. Millors paràmetres**: kernel:sigmoid i C:10
- **2. Precisió**: 49.6%

Si ens fixam, veim que surt la mitja de precisió entre el HOG y el Canny sumats.

4.3 Selecció model

Després d'haver provat diferentes combinacions, ens quedarem amb la que millor precisió ha tingut, en aquest cas, la que utilitza **HOG** individualment. Anem a veure la configuració que segueix aquest model:

Parameter	***Value***
C	1
Kernel	Poly
Degree	3
Gamma	Scale
Coef0	0.0
Tolerance	1e-3

Aquesta sirà el nostre model final que utilitzarem en cas de voler classificar imatges a diferents paisatges.

5 Anàlisi resultats

Els resultats són gaire prometedors i ens han demostrat quines tècniques funcionen millor per a cada cosa. Per exemple, podem afirmar que les nostres dades no eren linealment separables i, per tant, havíem d'utilitzar un kernel **polynomic**. L'utilització d'altres no era viable i justificam l'actual per a les dades que tenim.

Respecte l'altre punt important, la feature estrella ha estat el **HOG**, ja que aquesta detecta les direccions i orientacions dels cantons (edges). D'aquesta manera, podem definir formes i aparences que no podem fer amb altres features.

5.1 Conclusions finals

Finalment, anem a veure quines conclusions podem extreure del nostre desenvolupament i tota el seu procés:

- **1. Molta experimentació**: cal destacar que sense una base del coneixement de les característiques i com funcionen per dintre és difícil escollir-ne unes. A base de documentació i anar fent moltes proves hem pogut trobar una solució mitjanament òptima. Aquest treball ha estat molt interessant per a poder veure quines features diferents hi ha i per un primer contacte amb visió per computador, amb un dataset senzill. D'aquesta manera, no hem necessitat hardware excessivament potent per a resoldre aquest problema.
- **2. Bona precisió amb 'simplicitat'**: un altre factor a tenir en compte és que malgrat només haver aconseguit un 60% de precisió, no hem hagut de realitzar moltes transformacions i les dades estaven mitjanament netes, tan sols hem fet resize i convertit a *GrayScale*.

5.2 Millores per a possible futur model

Coses que faríem per a millorar el rendiment d'un altre possible model:

- **1. Augmentar recursos de computació**: el fet d'augmentar els nostres recursos ens siria molt útil a la hora de testejar més combinacions diferents, ja que hi podria haver combinacions més efectives. Tot i això, s'hauria de veure si l'augment de precisió val la pena per tot el cost en maquinària. Un inconvenient és que la llibreria Sklearn no té suport per a GPU, fet que ens faria tirar tot de processador.
- **2. Aplicació d'altres tècniques**: una tècnica que ens siria molt útil és el fet d'utilitzar **data augmentation**. Si tenim més dades evitam que es produesqui *overfitting* i augmentaríem la precisió ja que totes les imatges no sempre vendran donades amb mateix angle, lluminositat etc. Aquests són factors de les imatges que poden canviar i afectar la nostra possible predicció.
- **3. Augmentar nombre combinacions a testejar**: relacionat amb la primera premissa, si testejam un major nombre de combinacions diferents trobarem un model que obtengui una millor precisió.

6 Feina per parelles

Al fer feina conjunta cada sessió per Discord, hem tingut una dinàmica de feina molt equilibrada i tots dos hem tocat equitativament el codi.

7 Bibliografia

Aquí totes les fonts de dades que hem utilitzat:

- Sklearn
- ChatGPT 3.5