

```
In [25]: %%html  
<link rel= "stylesheet" type= "text/css" href= "style.css">
```

# PRÀCTICA 1

# EL PROCÉS DE L'APRENENTATGE

# AUTOMÀTIC

## Classificador GatIGos amb Màquines de Vectors de Suport(SVM)

Aprenentatge automàtic 2024-25



Antoni Frau Gordiola - 43482642S

### Primeres passes, cerca del millor model

#### Funcions auxiliars

Aquests són els primers *imports* i funcions auxiliars per obtenció d'anotacions, dibuixat d'imatges, etc. Part d'aquest material ha estat proporcionat pels professors de l'assignatura.

```
In [2]: import os  
import numpy as np  
import random  
import xml.etree.ElementTree as etree  
from skimage.io import imread  
from skimage.transform import resize  
import matplotlib.pyplot as plt  
from skimage.feature import hog  
from sklearn.model_selection import train_test_split
```

```
In [3]: # Parseja el fitxer xml i recupera la informació necessària per trobar la cara de l'animal  
#  
def extract_xml_annotation(filename):  
    """Parse the xml file  
    :param filename: str  
    :return annotation: diccionari  
    """  
    z = etree.parse(filename)  
    objects = z.findall('.//object')  
    size = (int(float(z.find('.//width').text)), int(float(z.find('.//height').text)))  
    dds = []  
    for obj in objects:  
        dds.append(obj.find('name').text)  
        dds.append([int(float(obj.find('bndbox/xmin').text)),  
                   int(float(obj.find('bndbox/ymin').text)),  
                   int(float(obj.find('bndbox/xmax').text)),  
                   int(float(obj.find('bndbox/ymax').text))])  
  
    return {'size': size, 'informacio': dds}
```

```
In [4]: # Selecciona la cara de l'animal i la transforma a la mida indicat al paràmetre mida_desti  
def retall_normalitzat(imatge, dades, mida_desti=(64,64)):  
    """  
    Extreu la regió de la cara (ROI) i retorna una nova imatge de la mida_desti
```

```
:param imatge: imatge que conté un animal
:param dades: diccionari extret del xml
:mida_desti: tupla que conté la mida que obtindrà la cara de l'animal
"""
x, y, ample, alt = dades['informacio'][1]
rettall = np.copy(imatge[:alt, x:ample])
return resize(retall, mida_desti)
```

```
In [5]: def obtenir_dades(carpeta_imatges, carpeta_anotacions, mida=(64, 64)):
    """Genera la col·lecció de cares d'animals i les corresponents etiquetes
    :param carpeta_imatges: string amb el path a la carpeta d'imatges
    :param carpeta_anotacions: string amb el path a la carpeta d'anotacions
    :param mida: tupla que conté la mida que obtindrà la cara de l'animal
    :return:
        images: numpy array 3D amb la col·lecció de cares
        etiquetes: llista binaria 0 si l'animal és un moix 1 en cas contrari
    """
    n_elements = len([entry for entry in os.listdir(carpeta_imatges) if os.path.isfile(os.path.join(carpeta_imatges, entry))])
    # Una matriu 3D: mida x mida x nombre d'imatges
    imatges = np.zeros((mida[0], mida[1], n_elements), dtype=np.float16)
    # Una Llista d'etiquetes
    etiquetes = [0] * n_elements

    # Recorre els elements de les dues carpetes: llegeix una imatge i obté la informació interessant del xml
    with os.scandir(carpeta_imatges) as elements:
        for idx, element in enumerate(elements):
            nom = element.name.split(".")
            nom_fitxer = nom[0] + ".xml"
            imatge = imread(carpeta_imatges + os.sep + element.name, as_gray=True)
            anotacions = extract_xml_annotation(carpeta_anotacions + os.sep + nom_fitxer)

            cara_animal = retall_normalitzat(imatge, anotacions, mida)
            tipus_animal = anotacions["informacio"][0]

            imatges[:, :, idx] = cara_animal
            etiquetes[idx] = 0 if tipus_animal == "cat" else 1

    return imatges, etiquetes
```

```
In [6]: def obtenirHOG(imatges, hogParams):
    """
    Genera les característiques i imatges dels gradients del corresponent histogram of gaussians
    :param imatges: col·lecció d'imatges de les quals es vol generar el vector de característiques
    :param hogParams: col·lecció dels 3 paràmetres de hog(pixels_per_cell, orientations, cells_per_block)
    :return:
        descriptors: descriptors de hog de les imatges
        imatges_hog: imatges que representen els gradients
    """

    pixels_bloc = (hogParams[0], hogParams[0])
    orientacions = hogParams[1]
    blocs = (hogParams[2], hogParams[2])

    descriptors = []
    imatges_hog = []

    for idx in range(imatges.shape[0]):

        imatge = imatges[idx, :, :]
        descriptor, imatge_hog = hog(imatge,
                                       orientations=hogParams[1],
                                       pixels_per_cell=(hogParams[0], hogParams[0]),
                                       cells_per_block=(hogParams[2], hogParams[2]),
                                       block_norm='L2-Hys',
                                       visualize=True,
                                       transform_sqrt=True)

        descriptors.append(descriptor)
        imatges_hog.append(imatge_hog)

    descriptors = np.array(descriptors)
    imatges_hog = np.array(imatges_hog)

    return descriptors, imatges_hog
```

```
pixels_per_cell=(params[0], params[0]),
cells_per_block=(params[2], params[2]),
block_norm='L2-Hys',
visualize=True,
transform_sqrt=True)

axes[idx2*1].imshow(imatge_hog, cmap='gray')
axes[idx2*1].set_title(f"len(descriptors) {params}")
axes[idx2*1].axis('off')

.tight_layout()
.show()
```

```
In [8]: def showResults(n_imatges, fig_size, idx, imatges, imatges_hog, prediccio, real):
    """
    Mostra una fila d'imatges i imatges de HoG amb la predicció i valor real al damunt. Mostra n_imatges consecutives a partir de idx
    :param n_imatges: nombre d'imatges de la fila
    :param fig_size: regeix la mida de les imatges
    :param idx: index dins la llista d'imatges, imatges de HoG, prediccions i reals a partir del qual es mostra la fila
    :param imatges: imatges del qual s'extreuen les que es visualitzen al damunt
    :param imatges_hog: imatges del qual s'extreuen les que es visualitzen al davall
    :param prediccio: llistat de resultats de predicció, que es situen al damunt de les imatges de la fila
    :param real: llistat de resultats reals, que es situen al damunt de les imatges de la fila
    """

    fig, axes = plt.subplots(2, n_imatges, figsize=(fig_size, 4.5))

    for idx, i in enumerate(range(idx, idx + n_imatges)):

        axes[0, idx].imshow(imatges[i,:,:], cmap='gray')
        axes[0, idx].axis('off')
        axes[0, idx].text(0.5, 1.2, f"Real: {'moix' if real[i] == 0 else 'ca'}", ha='center', transform=axes[0, idx].transAxes)
        axes[0, idx].text(0.5, 1.05, f"Predic: {'moix' if prediccio[i] == 0 else 'ca'}", ha='center', transform=axes[0, idx].transAxes)

        axes[1, idx].imshow(imatges_hog[i,:,:], cmap='gray')
        axes[1, idx].axis('off')

    plt.subplots_adjust(wspace=0, hspace=0)
    plt.show()
```

## Obtenció del conjunt de dades

Primer de tot es carreguen les imatges i anotacions corresponents, i amb una mida adequada (es considera que  $128 \times 128$  és suficient per interpretar-ho tant el model com l'humà).

El que es fa abans d'entrenar el model final, és cercar els millors paràmetres de les configuracions pels models amb **linear**, **RBF** i **poly**. S'avança que aquests seran els models amb els que es realitzarà la pràctica. Per triar aquests millors paràmetres es farà la cerca exhaustiva amb **k-fold** i **GridSearch**, i per tant el més raonable és cercar la millor configuració amb un subconjunt de les dades inicials. S'ha fet una cerca amb aproximadament el **20%** del total, i per fer aquesta separació s'ha usat 'train\_test\_split'. D'aquesta forma es genera 'sub\_imatges', que representarà el subconjunt de les dades totals per a la cerca dels paràmetres. Es destaca que sempre que s'usi aquesta terminologia(**sub**) fa referència al subconjunt del 20%.

```
In [9]: carpeta_images = "gatigos/images" # NO ES POT MODIFICAR
carpeta_anotaciones = "gatigos/annotations" # NO ES POT MODIFICAR
mida = (128, 128)
imatges, etiquetes = obtener_dades(carpeta_images, carpeta_anotaciones, mida)
```

Nota: Per poder dividir i tractar bé les imatges es transposen amb np.transpose

```
In [10]: imatges = np.transpose(imatges, (2, 0, 1))
_, sub_imatges, _, sub_etiquetes = train_test_split(imatges, etiquetes, test_size=0.2, random_state=42)
```

Recerca i estudi de HoG

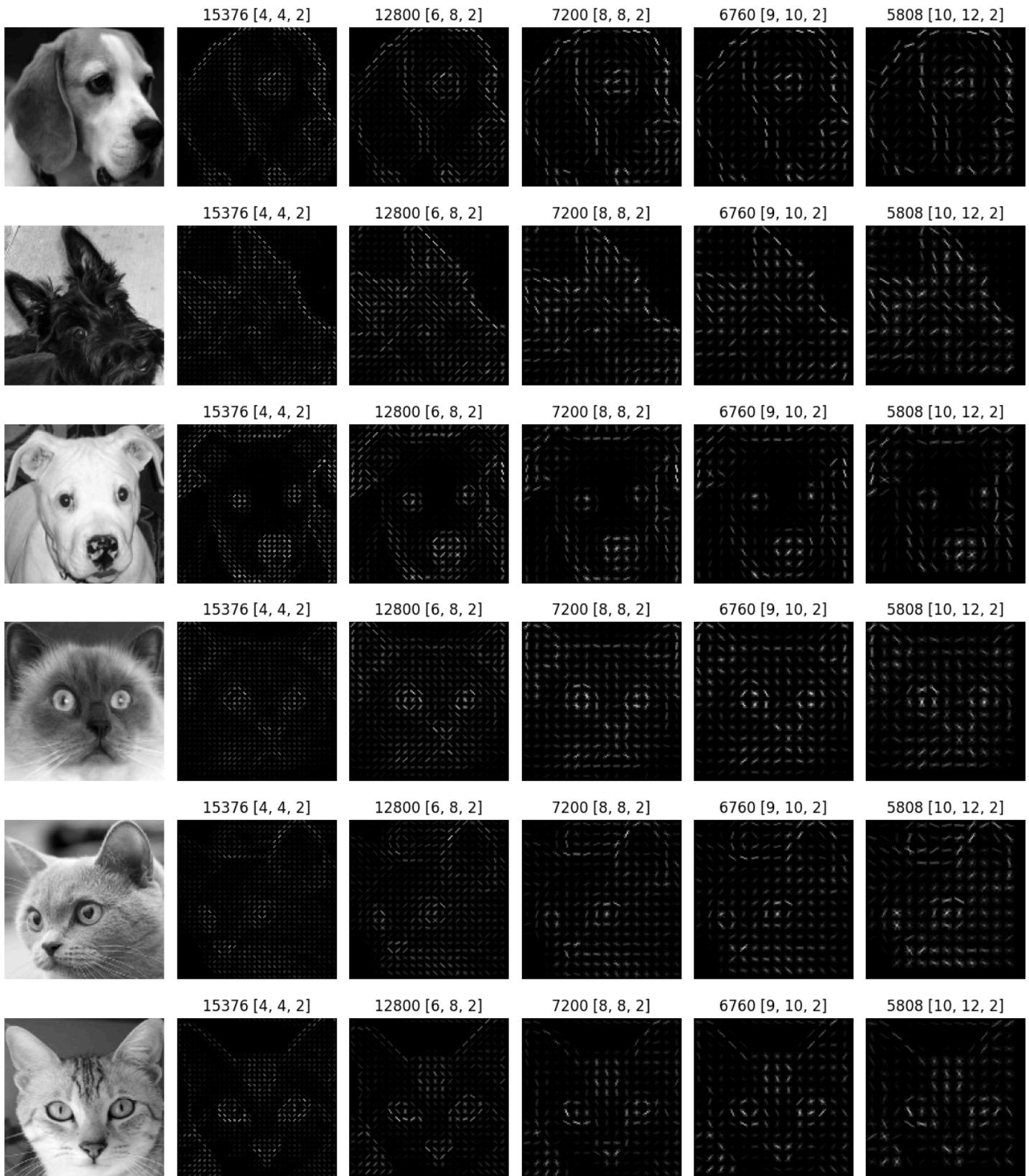
El següent, ja que com s'ha explicat a l'assignatura l'avenç amb l'ús de característiques per entrenar els models d'intel·ligència artificial, deriva en usar altres característiques que no siguin els mateixos pixels de les imatges. Una altra forma és usar els descriptors del *Histogram of Gaussians*(HoG), molt útil per entendre les voreres i formes de les imatges. I aquest és el seqüent pas, entendre com obtenir bons descriptors de HoG.

Una forma de veure-ho, és amb la funció auxiliar `showHoGVariations` implementada, que donada una imatge mostra representacions de HoG segons els llistats de les característiques que també es passen per paràmetre. A continuació es processaran 6 imatges de prova amb els paràmetres de HoG. Totes les imatges de prova visualitzades al llarg del document s'han triat a base de proves aleatòries, obtinguent resultats que considero representatius (per la forma dels animals, quantitat i qualitat de les dades, etc.).

També, com a títol de les imatges, s'inca el nombre de descriptors que resulten (un procés de HoG més el·laborat dona més descriptors) i els paràmetres del HoG corresponent. En aquest ordre, es mostren les dades:

- Nombre de píxels per cel·la
  - Nombre discret d'orientacions
  - Nombre de cel·les per bloc

```
#randomIdx = random.randint(0, len(imatges)-1)    Línia usada per extreure bones mostres aleatòries  
showHoGVariations(imatges[i, :, :], hog_params_prova, size_imatge)
```



```
In [12]: hog_parametres = [8, 8, 2]
```

Vists els resultats a les imatges, visualment i en quantitat de característiques té sentit triar els paràmetres de la tercera columna de HoG. Amb aquest, que usa `pixels_per_cell=8`, `orientations=8` i `cells_per_block=2`, es veu uns gradients prou definits i representatius de la imatge original. A més, la quantitat de descriptors/característiques és de 7200, el qual és bastant reduït, comparat amb les 16.384 característiques que s'usarien amb les imatges originals de 128x128 pixels.

També és clar que es podria entrenar el model amb diferents sortides de HoG i cercar el millor resultat, però no s'ha considerat a la pràctica.

Ara el següent pas és, amb els paràmetres triats, obtenir les característiques/descriptors HoG de les imatges per entrenar i estudiar els models.

Nota: El codi següent s'ha usat durant el desenvolupament de la pràctica, principalment per tenir a mà les característiques i alleugerir l'execució. Amb un valor de `accio = 0` té un funcionament convencional.

```
In [13]: # guarda o llegeix les característiques generades a un fitxer  
# format: saves/midaImatges_parametresHoG_nImatges/idx.npz
```

```
# 0 = calcular / 1 = guardar / 2 = calcular i guardar / 3 = llegir
accio = 3
hog_text = '_'.join(map(str, hog_parametres))
hog_text = f'{hog_parametres}'

ruta_guardat = f'saves/{mida[0]}_{hog_text}_{imatges.shape[0]}'
os.makedirs(ruta_guardat, exist_ok=True)

if accio == 0:
    print(f'L\'operació es faria a la ruta: {ruta_guardat}, però les dades es calculen')
    descriptors, imatges_hog = obtenirHoG(imatges, hog_parametres)
elif accio == 1 or accio == 2:
    if accio == 2:
        descriptors, imatges_hog = obtenirHoG(imatges, hog_parametres)

    ruta_guardat = f'{ruta_guardat}/{sum(1 for entry in os.listdir(ruta_guardat))+1}.npz'
    print(f'L\'escritura es fa a la ruta: {ruta_guardat}')
    np.savez(ruta_guardat,
              caract=descriptors,
              imgs_hog=imatges_hog)
elif accio == 3:
    ruta_guardat = f'{ruta_guardat}/{sum(1 for entry in os.listdir(ruta_guardat))}.npz'
    print(f'La lectura es fa a la ruta: {ruta_guardat}')
    dades = np.load(ruta_guardat)
    descriptors = dades['caract']
    imatges_hog = dades['imgs_hog']
```

La lectura es fa a la ruta: saves/128\_[8, 8, 2]\_3686/3.npz

In [14]: # també es separen les dades extretes de HoG, útil per a l'entrenament i visualització de les característiques  
\_, sub\_descriptors, \_, sub\_imatges\_hog = train\_test\_split(descriptors, imatges\_hog, test\_size=0.2, random\_state=42)

## Models de classificació i hiperparàmetres

### Quins paràmetres triar?

Una vegada es tenen totes les dades per fer l'entrenament i cerca dels millors paràmetres amb k-fold i GridSearch, es pot començar amb el procés. Primer fent les corresponents divisions en dades de train i test, l'estandardització de les dades, triar els diccionaris de cerca de paràmetres i, finalment, entrenar els subconjunts de les imatges per comparar quins hiperparàmetres són més efectius.

Just seguit es troben els *imports* corresponents, necessaris per a totes les funcions que s'usaran, principalment relacionades amb contingut de `sklearn`.

In [15]:

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score, classification_report
from scipy.spatial import distance_matrix
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.model_selection import GridSearchCV
from IPython.display import display, Markdown
```

Aquí es fa la divisió de les dades d'entrada en train i test. S'usa també `train_test_split`, i he decidit posar un **67%** en mostres d'entrenament i **33%** en test. Les nomenclatures segueixen la norma anterior d'usar `sub` per indicar que és el subconjunt del 20%, i a més, train i test per indicar que pertanyen al conjunt de train i test respectivament.

A més, de la següent forma també se separen les imatges visuals i de hog amb el mateix criteri. Això es fa per poder tenir resultats visuals més endavant.

In [16]:

```
sub_X_train, sub_X_test, sub_y_train, sub_y_test = train_test_split(sub_descriptors, sub_etiquetes, test_size=0.33, random_state=42)
sub_imatges_train, sub_imatges_test, sub_imghog_train, sub_imghog_test = train_test_split(sub_imatges, sub_imatges_hog, test_size=0.33, random_state=42)
```

Aquí s'estandarditzen les dades amb un `StandardScaler`

In [17]:

```
scaler = StandardScaler() #MinMaxScaler()
sub_X_train_transformed = scaler.fit_transform(sub_X_train)
sub_X_test_transformed = scaler.transform(sub_X_test)
```

Aquí es trien els diferents diccionaris de paràmetres per a la cerca del millor model de classificació amb SVC. L'estructura és una llista amb els tres diccionaris corresponents als models amb ell kernels `linear`, `rbf` i `poly` respectivament, separats en 3 línies diferents.

Quins són els motius de l'elecció dels paràmetres i dels valors? Per a cada kernel:

### Linear

$$K_{\text{linear}}(x, z) = x \cdot z$$

### C

Aquest és l'únic que controla la variació d'aquest kernel, i representa la penalització i, per tant, la correcció del model. Això vol dir que amb valors massa alts ajusta més els errors i, en conseqüència, pot provocar *overfitting*. Lògicament, amb valors menors no ajusta tant a l'error i, per tant, generalitza més. I també per aquestes raons és important no tenir valors massa grans ni massa petits. En general, per aquest problema, com l'overfitting seria un problema més greu (el conjunt d'imatges no és tan gran i les diferències entre cans i moixos poden ser molt poques) he triat un rang de valors més petits com 0.0001 fins a 1000.

### RBF / Gaussià

Aquest genera un model que ja no és lineal, i cal destacar que és el kernel per defecte de `SVC`.

$$K_{\text{RBF}}(x, z) = \exp(-\gamma \|x - z\|^2)$$

## C

Aquesta C comparteix el significat amb el kernel linear, ja que també representa la penalització. Els canvis més notables venen, perquè aquest no és l'únic paràmetre que es pot modificar, i per tant per reduir les exponencials combinacions he rebaixat el rang de possibles valors. No fan falta valors tan extrems, i per això els valors descartats han estat el més petit i el més gran, en relació amb l'anterior.

### Gamma

Aquest paràmetre apareix en els kernels no lineals, i regeix la forma en la qual influeixen les mostres en l'entrenament. Amb valors més alts, els punts, i els suports, tenen més importància i els límits de la classificació s'ajusten més a les dades. En canvi, amb valors més baixos permet generalitzar en la classificació. En aquest cas els valors triats van d'un rang més baix, del 1e-1 al 1e-5 (no passa per tots els valors intermedis), i incloent també *auto* i *scale*. Aquests dos darrers es calculen internament de forma que el primer s'ajusta al nombre de característiques  $\frac{1}{n\_caract}$  i el segon també i a més amb la seva variança  $\frac{1}{n\_caract \times x.var()}$ .

## Poly

$$K_{poly}(x, z) = (\gamma x \cdot z + \text{coef0})^2$$

## C

També es comparteix el paràmetre C com les dues anteriors vegades. Es pot destacar que també està reduït respecte al kernel lineal perquè hi ha altres paràmetres i les combinacions finals creixen exponencialment.

### Gamma

Aquest també es comparteix, aquest cas amb RBF, de forma que representa l'impacte de cada mostra dins la classificació. Aquí també s'han reduït els valors a usar, com en el valor 0.1, perquè és un extrem amb un valor bastant elevat, no molt útil quan se cerca un model que té unes característiques que no són sempre exactes per a tots els cans/moixos.

### Degree

Un nou paràmetre que afecta aquest tipus de kernel. Com és clar, regeix el grau del polinomi, i, com més elevat més complex i permet classificacions més variades. Valors comuns que es poden usar per no excedir-se són del 2 al 5 (amb 1 el model és lineal), encara que el 5 no s'ha usat per mantenir compromís amb el temps d'entrenament.

### Coef0

El darrer paràmetre és rellevant quan el model és d'aquest tipus, polinòmic. Com també es pot veure a la funció, representa el terme independent, constant i que també és característic de les funcions polinòmiques. Valors útils per estudiar el rendiment del model són del 0 (funció sense desplaçament) fins a realment qualsevol no molt exagerat. Jo concretament he afegit l'1 i el 10.

```
In [18]: kernels = ["linear", "rbf", "poly"]
parameters_set = [{"C": [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]},
                  {"C": [0.001, 0.01, 0.1, 1, 10, 100], 'gamma': ['scale', 'auto', 0.1, 0.01, 0.001, 0.0001]},
                  {"C": [0.01, 0.1, 1, 10], 'gamma': ['scale', 'auto', 0.01, 0.001, 0.0001], 'degree': [2, 3, 4], 'coef0': [0, 1, 10]}]
```

## Triar la millor configuració de paràmetres

Per fer la cerca, com ja s'ha comentat anteriorment s'usa `GridSearchCV`, on aquest es pot entrenar amb el model i el diccionari de paràmetres adequat. En acabar, es pot obtenir el millor estimador `best_estimator_`, que és la pròpia configuració del model de classificació.

Per a tot el procés, es fa la cerca per als 3 kernels dins un bucle, i es van registrant els millors estimadors de cada model així com les seves puntuacions (demanades a l'enunciat de la pràctica) que indiquen el resultat de fer *k-fold cross-validation*. Aquestes mètriques són la precisió(**precision**), sensibilitat(**recall**) i especificitat(**F1**).

```
In [19]: best_estimators = [{}, {}, {}]
precisions = [0, 0, 0]
recalls = [0, 0, 0]
f1s = [0, 0, 0]

for idx, kernel in enumerate(kernels):
    svm = SVC(kernel=kernel)
    clf = GridSearchCV(svm, parameters_set[idx], cv=3)

    clf.fit(sub_X_train_transformed, sub_y_train)

    print(f"Millor model {kernel} amb: {clf.best_estimator_}")
    best_estimators[idx] = clf.best_estimator_
    resultats = clf.best_estimator_.predict(sub_X_test_transformed)

    precisions[idx] = precision_score(sub_y_test, resultats, average='weighted')
    recalls[idx] = recall_score(sub_y_test, resultats, average='weighted')
    f1s[idx] = f1_score(sub_y_test, resultats, average='weighted')

Millor model linear amb: SVC(C=0.0001, kernel='linear')
Millor model rbf amb: SVC(C=10, gamma=1e-05)
Millor model poly amb: SVC(C=0.01, coef0=10, gamma=0.0001, kernel='poly')
```

```
In [20]: taula_resultats = """
|          |          **Precisió** |          **Sensibilitat** |          **Especificitat** |
|-----|-----|-----|-----|
| **Model lineal** | {round(precisions[0], 3)} | {round(recalls[0], 3)} | {round(f1s[0], 3)} |
| **Model RBF** | {round(precisions[1], 3)} | {round(recalls[1], 3)} | {round(f1s[1], 3)} |
| **Model polinòmic** | {round(precisions[2], 3)} | {round(recalls[2], 3)} | {round(f1s[2], 3)} |
"""

display(Markdown(taula_resultats))
```

	Precisió	Sensitivitat	Especifitat
<b>Model lineal</b>	0.906	0.906	0.906
<b>Model RBF</b>	0.901	0.902	0.901
<b>Model plonòmic</b>	0.893	0.893	0.893

De forma que a aquesta taula queden representats tots indicadors de l'eficiència del model, almanco amb el subconjunt inicial de dades. Puc indicar que els indicadors estan calculats en funció dels pesos de les mostres, ja que no hi ha la mateixa quantitat de mostres d'una classe que la de l'altra. Aquest càlcul implica que els resultats són reals però suavitzats, i per tant la precisió, sensitivitat i especificitat tenen valors molt pròxims. Si no, puc indicar que també es podria veure el rendiment amb `classification_report`, que aporta moltes dades d'aquest tipus d'indicacions, com la precisió, sensitivitat i especificitat per pesos, l'`accuracy`, etc. No s'ha implementat perquè no ho he considerat necessari, és suficient indicar aquests resultats per separat.

A continuació es pot treure el millor model, que es considera que és el que té l'F1 més alt, ja que depèn dels altres dos indicadors (és una mitja harmònica). Es veu que el que té millor rendiment, amb les dades triades (que poden variar com mida de les imatges, paràmetres de HoG, hiperparàmetres cercats i mode de k-fold cross-validation), és el model lineal.

```
In [21]: best_params = best_estimators[f1s.index(max(f1s))].get_params()
print('Els millors paràmetres pel model de classificació són: {best_estimators[f1s.index(max(f1s))]}')

Els millors paràmetres pel model de classificació són: SVC(C=0.0001, kernel='linear')
```

## Entrenament i test complet

Ja amb totes les dades d'entrenament, elegits els paràmetres de HoG per extreure els descriptors corresponents, i els hiperparàmetres del millor model amb el subconjunt de dades, es pot construir i testejar el model final.

A partir d'aquí es repeteix i s'unifica tot el procés anterior amb les dades corresponents. La diferència principal és que domés s'entrena i testeja un model (amb els paràmetres triats), i al final es recullen i es mostren els indicadors d'eficiència del model.

```
In [22]: X_train, X_test, y_train, y_test = train_test_split(descriptors, etiquetes, test_size=0.33, random_state=42)
imatges_train, imatges_test, imghog_train, imghog_test = train_test_split(imatges, imatges_hog, test_size=0.33, random_state=42)

scaler = StandardScaler()
X_train_transformed = scaler.fit_transform(X_train)
X_test_transformed = scaler.transform(X_test)

svc = SVC(**best_params)
svc.fit(X_train_transformed, y_train)
predicccio = svc.predict(X_test_transformed)

precisio = precision_score(y_test, predicccio, average='weighted')
recall = recall_score(y_test, predicccio, average='weighted')
f1 = f1_score(y_test, predicccio, average='weighted')

taulta_resultats = f"""
|           | **Precisió**      | **Sensitivitat** | **Especifitat** |
|-----|-----|-----|-----|
| **Model lineal** | {round(precisio, 3)} | {round(recall, 3)} | **{round(f1, 3)}** |
"""

display(Markdown(taula_resultats))
```

	Precisió	Sensitivitat	Especifitat
<b>Model lineal</b>	0.911	0.911	<b>0.911</b>

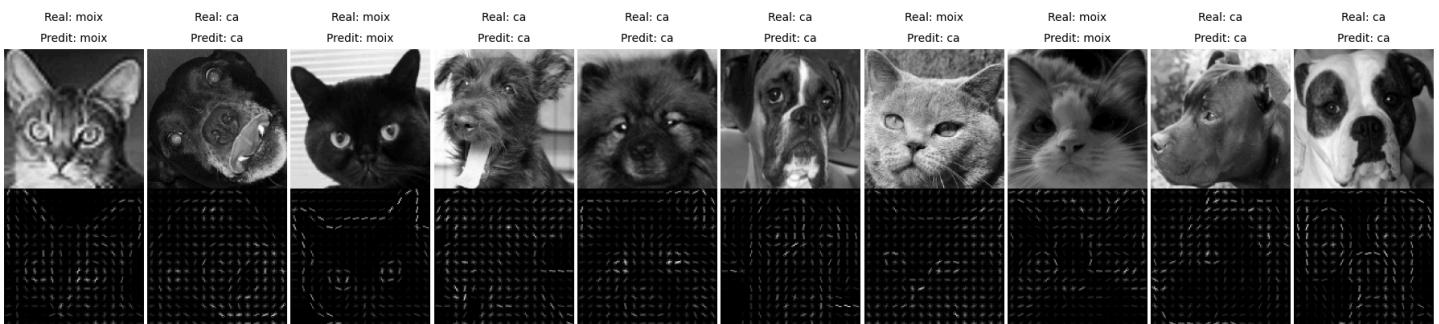
Finalment, es conclou que el rendiment és bastant acceptable, amb una precisió del **91%** i també bona especificitat.

Aquí he decidit mostrar algunes imatges del conjunt de test amb les seves prediccions i valors reals. També he afegit les imatges dels gradients de HoG per veure la seva representació. Exactament igual que amb la visualització a l'hora de triar paràmetres de HoG, les imatges que es mostren s'han cercat prèviament i deixat un subconjunt de 10 imatges amb resultats representatius:

```
In [23]: n_imatges = 10
idx = 697
fig_size = 23

#randomIdx = random.randint(0,imatges_test.shape[0]-n_imatges)    Línia usada per extreure bones mostres aleatòries

showResults(n_imatges, fig_size, idx, imatges_test, imghog_test, predicccio, y_test)
```



Vists aquests resultats, on s'agafen 10 imatges consecutives del conjunt de test, ja es pot veure la distribució amb majoritàriament més cans respecte a moixos, encara que hi haurà vegades on hi hagi més moixos concentrats. A part, es veuen ben definides les característiques de molts dels animals als seus gradients. Algunes imatges més "complicades" d'identificar poden ser (d'esquerra a dreta): la segona, quarta, quinta i sèptima. En general, el motiu és perquè al gradient hi ha molt de renou, molta variació en els contorns que no deixa veure bé la forma de l'animal. També pot afectar que aquest no estigui perfectament alineat amb la imatge, com pot ser el segon ca, un poc tort. Amb tot això, es pot entendre com a la setena imatge el model falla, i detecta un ca en lloc d'un moix.

## Dificultats i indicacions

Cal destacar primerament, la necessitat de descarregar tots els paquets per a executar el *notebook*. Els adjunt per si fa falta algun:

```
In [24]: #conda install numpy
#conda install -c conda-forge scikit-image
#conda install matplotlib
#conda install -c conda-forge scikit-learn
#conda install scipy
```

Més endavant, en el procés d'obtenir les imatges i les seves característiques (descriptors de HoG), no he tengut gaires problemes, ja que amb la referència a la *documentació de scikit*, he vist exemples útils. Com més endavant he pogut jugar i veure jo mateix resultats reals, fent proves amb imatges aleatories, tampoc he trobat problemes per triar els paràmetres de HoG. Es podria dir que aquí el problema és quedar conforme amb el nombre de descriptors per a entrenar el model (fer curt o passar-se). Clar que la millor forma seria fer proves entrenant amb diverses configuracions i veient els resultats, però finalment he considerat un nombre de descriptors i gradients visuals "a ull". També es pot destacar que la part de visualitzar imatges, tant les imatges del *dataset* com de hog, i l'ús de subplots de matplotlib, m'he ajudat de ChatGPT i la *documentació oficial*.

Seguint amb el procés, a la cerca dels millors paràmetres i configuracions per a cada model, el clar i principal problema era quins triar i en què basar-me. Al final, he investigat per la documentació aportada pel professorat de les *mètriques*. Com l'F1 és una mitja dels altres dos indicadors, he decidit usar aquest com a referència. Al darrer, la forma d'estudiar els paràmetres usats ha estat també amb proves i documentació de scikit, de forma que obtinguen uns resultats acceptables (aproximadament 90%) i considerant un temps d'entrenament també baix (aproximadament 2 minuts en els tres models) he trobat els paràmetres que consider més adients.

Finalment, en tenir-ho tot no ha estat molt complicat, ajuntar el millor estimador en F1 amb un nou model SVM i entrenar-ho amb tot el conjunt de dades. El resultat final és més alt que a l'entrenament, i, encara que no massa, ho considero bon resultat final. Com s'ha vist, també amb les imatges finals, que n'he decidit usar 10 perquè la ràtio de certesa es pot simplificar a 9:10 imatges.



En conclusió, aquesta ha estat la meva pràctica 1.