Tarea 2

Instrucciones

1. Selecciona 10 clases de la base de datos Caltech 10.
   https://data.caltech.edu/records/mzrjq-6wc02

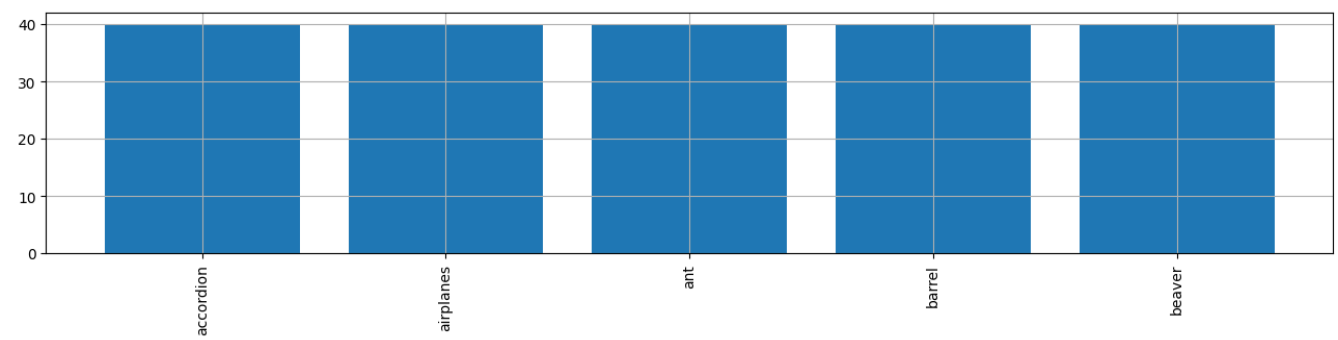
```
# Find the name of each class
base_path = "/content/drive/MyDrive/Colab Notebooks/Vision Computacional -semana2/Dia Martes/caltech-101"
class_names = listdir(base_path)[:5]
print("Num of classes:", len(class_names))
```

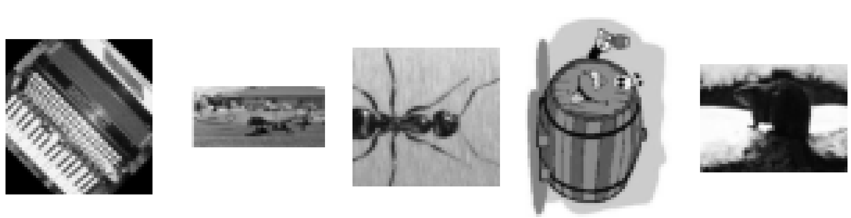
```
Num of classes: 5
```

2. Selecciona 40 imágenes por cada una de las clases: 400 imágenes en total.

```
# Load first 40 images from first class and a label for the class
X = []
Y = []
X_frec=[]
for clase in range(0,5):
    print(class_names[clase])
    file_names1 = [join(base_path, class_names[clase], f) for f in listdir(join(base_path, class_names[clase]))]
    X1 = np.array([resize(imread(f, as_gray='True'), (300, 200)) for f in file_names1[:40]])
    Y1 = np.full((len(X1)), clase)
    X_frec.append(len(X1))

    X.append(X1)  # Agregar los datos de la clase actual a la lista X
    Y.append(Y1)  # Agregar las etiquetas de la clase actual a la lista Y

X = np.concatenate(X)  # Convertir la lista de datos en un arreglo numpy
Y = np.concatenate(Y)  # Convertir la lista de etiquetas en un arreglo numpy

print(X.shape)
print(Y.shape)
```

HoG= (64,128)





3. Calcula los descriptores GIST, HoG, y SIFT+BoW para cada imagen.

- GIST

```
[ ]
# Compute GIST for each image
param = {"orientationsPerScale":np.array([8, 8, 8, 8]),
         "numberBlocks":[4, 4],
         "fc_prefilt":10,
         "boundaryExtension":10}
gist = GIST(param)

X_gist = np.array([gist._gist_extract(resize(img, (349, 352))) for img in X])
print(X_gist.shape)
```

```
(200, 512)
```

- HoG

```
# Compute HOG for each image
HOG = np.array([hog(img) for img in X])
print(HOG.shape)

```

```
(200, 6804)
```

- SIFT+BoW

```
def compute_SIFT(file_path):
    img = cv2.imread(file_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    sift = cv2.SIFT_create()
    _, desc = sift.detectAndCompute(gray, None)
    desc = np.clip(normalize(desc), 0.0, 0.2)
    return(normalize(desc))
```

```
from random import sample

train_SIFTS = []
for file_name in sample(names_train, len(names_train)):
    train_SIFTS.extend(compute_SIFT(file_name))
    if len(train_SIFTS) >= 50_000:
        break

train_SIFTS = np.array(train_SIFTS)
```

```
# Train a visual dictionary
from sklearn.cluster import KMeans

num_clusters = 500
kmeans = KMeans(n_clusters=num_clusters, n_init=10).fit(train_SIFTS)
print(f"Inertia: {kmeans.inertia_}")

```

```python
# Read an image, estimate its SIFT descriptors and BoW representation
def get_visual_words(file_path):
    SIFTS = compute_SIFT(file_path)
    v_words = kmeans.predict(SIFTS)
    return(v_words)
```

```python
# Compute the BOW representation for all the training set
BOW_train = np.zeros((len(names_train), num_clusters))
bins = np.arange(0, num_clusters+1)
for it_file, file_name in enumerate(names_train):
    v_words = get_visual_words(file_name)
    BOW_train[it_file], _ = np.histogram(v_words, bins, density=True)
```

```python
# Show all BOW representations sum up to one
plt.figure(figsize=(12, 3))
plt.plot(BOW_train.sum(axis=1))
plt.show()

print(BOW_train.shape)
print(y_train.shape)
```

4. Subdivide tus imágenes en sets de entramiento y prueba.

GIST

```python
# Split train and test sets
x_train, x_test, y_train, y_test = train_test_split(X_gist, Y, test_size=0.1)

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(180, 512)
(20, 512)
(180,)
(20,)
```

HoG

```python
# Split train and test sets
x_train, x_test, y_train, y_test = train_test_split(HOG, Y, test_size=0.1)

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(180, 6804)
(20, 6804)
(180,)
(20,)
```

SIFT+BoW

```python
# Split training and test sets
names_train, names_test, y_train, y_test = train_test_split(all_files_names, all_files_classes, test_size=0.1)

print(y_train.shape)
print(y_test.shape)
```

```
(180,)
(20,)
```

5. Entrena un clasificador por cada descriptor de imágenes (realiza GridSearch para encontrar los mejores hiperparámetros).

```
# Gradient Boosting **************************************************

# Define grid search parameters
hyperparams = {'learning_rate': [0.001, 0.01, 0.1],
               'n_estimators': [2, 5, 10],
               'max_depth': range(2, 7),
               'min_samples_split': range(2, 7),
               'min_samples_leaf': range(1, 6, 2),
               'max_features': [None, 'sqrt', 'log2']}
hyperparams
```

```
{'learning_rate': [0.001, 0.01, 0.1],
 'n_estimators': [2, 5, 10],
 'max_depth': range(2, 7),
 'min_samples_split': range(2, 7),
 'min_samples_leaf': range(1, 6, 2),
 'max_features': [None, 'sqrt', 'log2']}
```

GIST

```
# Create and train the classifiers with grid search
gs_model = GridSearchCV(GradientBoostingClassifier(), hyperparams, verbose=True, n_jobs=3)
gs_model.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 1215 candidates, totalling 6075 fits
▸ GridSearchCV
▸ estimator: GradientBoostingClassifier
  ▸ GradientBoostingClassifier
```

HoG

```
# Create and train the classifiers with grid search
gs_model = GridSearchCV(GradientBoostingClassifier(), hyperparams, cv=2, verbose=True, n_jobs=3)
gs_model.fit(x_train, y_train)
```

```
Fitting 2 folds for each of 2025 candidates, totalling 4050 fits
▸ GridSearchCV
▸ estimator: GradientBoostingClassifier
  ▸ GradientBoostingClassifier
```

SIFT+BoW

```
# Create and train the classifiers with grid search
gs_model = GridSearchCV(GradientBoostingClassifier(), hyperparams, verbose=True, n_jobs=3)
gs_model.fit(BOW_train, y_train)
```

```
Fitting 5 folds for each of 2025 candidates, totalling 10125 fits
▸ GridSearchCV
▸ estimator: GradientBoostingClassifier
  ▸ GradientBoostingClassifier
```

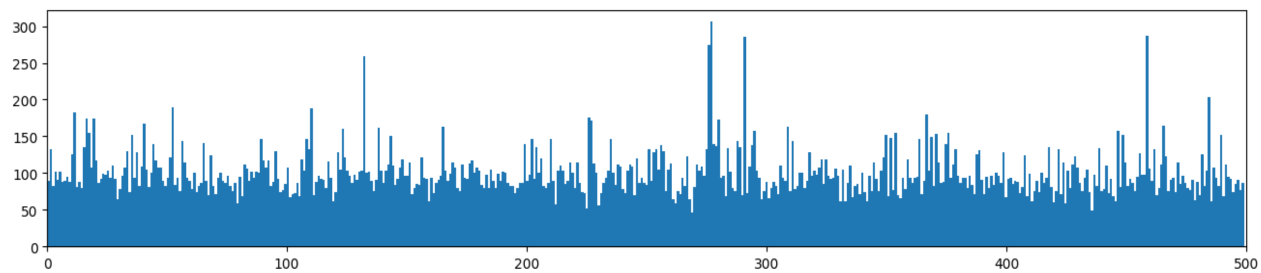
6. Reporta el mejor clasificador para cada combinación. Puedes utilizar la siguiente tabla como ejemplo.

	Train Acc.	Val Acc.	Test Acc.	Tiempo de entrenamiento
GIST	1.000	0.861	0.800	23 min
HoG	1.000	0.800	0.750	1h,45min
SIFT+BoW	1.000	0.672	0.550	26 min

6. Incluye el tamaño de la bolsa de palabras visuales de modelo BoW.

```python
num_clusters = 500
kmeans = KMeans(n_clusters=num_clusters, n_init=10).fit(train_SIFTS)
print(f"Inertia: {kmeans.inertia_}")

# Show frequency distribution of words
plt.figure(figsize=(16, 3))
plt.hist(kmeans.labels_, num_clusters)
plt.xlim(0, num_clusters)
plt.show()
```

```
Inertia: 12452.669921875
```



7. Reporta la matriz de confusión. Únicamente para el mejor modelo de todas las combinaciones probadas.

GIST