RECONOCIMIENTO

Módulo Machine Learning (e6.py)

- Importar datos de Titere/Weka
- Clasificador SVM / Bayesiano
- Clasificador Decision Tree / Neural Network (MLP)
- Validación: Matriz de Confusión, Precision/Recall/F-Score
  TP-Rate, FP-Rate
- Librerías: **opencv** / **scikit-learn**

https://docs.opencv.org/4.5.5/dd/ded/group__ml.html

https://docs.opencv.org/4.5.5/d6/de2/tutorial_py_table_of_contents_ml.html

https://scikit-learn.org/stable/user_guide.html

Reconocimiento de imágenes

- Programa base: **e6.py**
- Leer datos entrenamiento Titere/Weka: módulo **titere.py**

```
from scipy.io import arff

def readWekaData(file, attributes, labelColum, labelRange=None):
    data, metadata = arff.loadarff( file )                # Load weka data file (.arff)
    attrNames = metadata.names()
    attrTypes = metadata.types()

    # extract features and convert the structured array (list of tuples) to a standard np.array
    # in the structured array, data columns can be indexed by feature name
    dataMat = np.empty((len(data), len(attributes)), dtype=np.float32)
    for idx, item in enumerate(attributes):
        dataMat[:, idx] = data[item].astype(np.float32)

    # Label vector string and class number (starting at 1)
    labelName = data[labelColum].astype(str)
    if labelRange is None:
          labelRange = set(labelName)     # range of labels
          labelRange = sorted(labelRange)  # sort label set -> list

    label = np.zeros(len(labelName), dtype=int)    # init  label vector to zero
    for idx, item in enumerate(labelRange):
        label[labelName==item] = idx+1

    return dataMat, label, labelName, labelRange
```

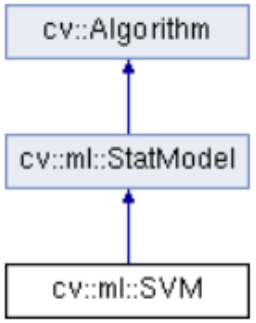
Reconocimiento de imágenes

- Programa base: **e6.py**

```
import titere

DATA_FILE = 'train.arff'          # default data file
TEST_FILE = 'test.arff'           # default data file
CLASSIFIER = 'svm'                # svm | bayes | dtree | mlp

# import titere
attributes = ('Compacidad', 'Excentricidad',  'Rel_Invar_1', 'Rel_Invar_2')
labelColum = 'Pieza'
trainData, trainLabels, trainLabelName, labelRange = titere.readWekaData(DATA_FILE , attributes, labelColum)
testData, testLabels, testLabelName, labelRange = titere.readWekaData(TEST_FILE, attributes, labelColum,
                                                                      labelRange)

# display data
print(f"{attributes} | {labelColum} (Num/Label)")
print(np.column_stack((trainData, trainLabels, trainLabelsName)))
print(f"{labelRange=}")
```

Reconocimiento de imágenes

- Programa base: **e6.py** (módulo *machine learning*)
  - Clase: **cv.ml_SVM** (Support Vector Machine)
    - cv.ml.**SVM_create**() → svm (cv.ml_SVM)



  - Métodos cv.ml.SVM:
    - cv.ml_SVM.**setType**(**val**) # cv.ml.**SVM_C_SVC** | **SVM_NU_SVC** | **SVM_ONE_CLASS** | **SVM_EPS_SVR** | **SVM_NU_SVR**
    - cv.ml_SVM.**setKernel**(**kernelType**). # cv.ml.**SVM_LINEAR** | **SVM_POLY** | **SVM_RBF** | **SVM_SIGMOID** | **SVM_CHI2** | **SVM_INTER** | **SVM_CUSTOM**

$K(x_i, x_j) = x_i^T x_j$ lineal

$K(x_i, x_j) = (\gamma x_i^T x_j + coef0)^{degree}, \gamma > 0$ Poly

$K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}, \gamma > 0$ RBF

$K(x_i, x_j) = \tanh(\gamma x_i^T x_j + coef0)$ Sigmoid

    - cv.ml_SVM.**setGamma**(**va**) # POLY | RBF | SIGMOID | CHI2
    - cv.ml_SVM.**setCoref0**(**va**) # POLY | SIGMOID
    - cv.ml_SVM.**setDegree**(**va**) # POLY

    - cv.ml_SVM.**setC**(**va**) # for C_SVC, classifier. penalty multiplier for outliers (def. 1.0)
    - cv.ml_SVM.**setNu**(**va**) # for NU_SVC, classifier [0,1] the larger the value, the smoother the decision boundary (Def. 0.5)
    - cv.ml_SVM.**setP**(**va**) # for EPS_SVR, epsilon in loss function (Regression)

    - cv.ml_SVM.**getSupportVectors**() → retval ndarray(nsv, nfeatures, np.float32)

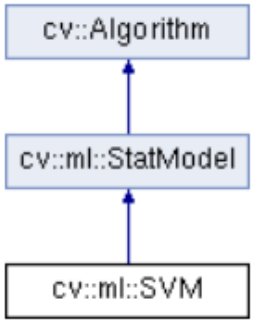
Reconocimiento de imágenes

- Programa base: **e6.py** (módulo *machine learning*)
  - Clase: **cv.ml_SVM** (Support Vector Machine)
    - cv.ml.**SVM_create**() → svm (cv.ml_SVM)
  - Métodos cv.ml.SVM:
    - cv.ml_SVM.**setTermCriteria**(**val**) #Finalización del algoritmo de optimización de entrenamiento

      val: tuple (criteria, maxCount, epsilon)

      criteria: cv.**TermCriteria_MAX_ITER** | cv.**TermCriteria_EPS**
      - **MAX_ITER:** número máximo de iteraciones
      - **EPS:** error máximo, diferencia en la función de coste entre dos iteraciones

Reconocimiento de imágenes



- Programa base: **e6.py** (módulo *machine learning*)
  - Clase base (train/predict) : **cv.ml_StatModel**
    - cv.ml_StatModel.**train**(**samples**, **layout**, **responses**) → retval (bool)
    - cv.ml_StatModel.**train**(**trainData**[, **flags**]) → retval (bool)
      - **samples:** The input samples, floating-point matrix (np.float32 / cv.CV_32F)
      - **layout:** cv.ml.**ROW_SAMPLE** , cv.ml.**COL_SAMPLE**
      - **responses:** vector of responses associated with the training samples
      - **trainData:** training data created with cv.ml.**TrainData_create**
      - **retval**: (bool)
    - cv.ml_StatModel.**predict**(**samples** [, **results** [, **flags**]]) → retval, results
      - **samples:** The input samples, floating-point ndarray(m,nc)
      - **results:** output matrix of results. ndarray(m,1) (optional for single sample)
      - **retval**: (float) prediction for single sample
  - Clase cv::Algorithm
    - (Método) cv.Algorithm.**save**(filename) Guarda el modelo entrenado (JSON/YAML)
    - (Método) cv.ml.SVM.**load**(filename) → svm (cv.ml_SVM) lee el modelo entrenado
    - (Funcion) cv.ml.**SVM_load**(filename) → svm (cv.ml_SVM) lee el modelo entrenado

Reconocimiento de imágenes

- Programa base: **e6.py** **SVM** (módulo machine learning)

```
# create classifier
svm = cv.ml.SVM_create()

# Configure SVM Classifier

# SVM Type: cv.ml.SVM_C_SVC | SVM_NU_SVC | SVM_ONE_CLASS | SVM_EPS_SVR | SVM_NU_SVR
svm.setType(cv.ml.SVM_C_SVC)

# Kernel type: cv.ml.SVM_LINEAR | SVM_POLY | SVM_RBF | SVM_SIGMOID | SVM_CHI2 | SVM_INTER | SVM_CUSTOM
svm.setKernel(cv.ml.SVM_RBF)
svm.setC(1.0)        # classifier. penalty multiplier for outliers
svm.setGamma(1.0)    # RBF parameter
svm.setTermCriteria( (cv.TermCriteria_MAX_ITER | cv.TermCriteria_EPS, 10000, 1e-6) )

# Train SVM
svm.train(trainData, cv.ml.ROW_SAMPLE, trainLabels)
svm.save("SVM.json")              # Save trained classifier

retval, prediction = svm.predict(testData)                    # Test SVM (Prediction)

prediction = prediction[:, 0].astype(int)      # convert to integer vector
print(f"Predicted: ", prediction)
print(f"Real:      ", testLabels)

titere.showPerformance(testLabels,  prediction, labelRange)
sv = svm.getSupportVectors()
print(f"Support Vectors: ", len(sv))
```

Reconocimiento de imágenes

- Matriz de Confusión:

```
def showPerformance(label, prediction, labelRange):
    nc = len(labelRange)
    confusionMat = np.zeros((nc, nc), dtype=np.int32)

    # Confusion matrix: (rows: actual class, cols: predicted class)
    for idx,predictedClass in enumerate(prediction):
        actualClass = label[idx]
        confusionMat[actualClass-1, predictedClass-1] += 1

    test_count = confusionMat.sum()
    accuracy = confusionMat.trace(dtype=np.float32) / test_count  # sum diagonal elements

    print("\nConfusion Matrix:", confusionMat)
    print("\n          F-Score | Precision | Recall/TPRate |  FPRate  ")
    print("-----------------------------------------------------")
    for r in range(nc):
        recall = 0.0; precision = 0.0; fScore = 0.0
        if confusionMat[r,:].sum() != 0:  recall = float(confusionMat[r, r]) / confusionMat[r,:].sum()
        if confusionMat[:,r].sum() != 0:  precision = float(confusionMat[r, r]) / confusionMat[:,r].sum()
        if (recall + precision) != 0: fScore = 2 * recall*precision / (recall + precision)
        fpRateNum = 0.0;     fpRateDen = 0.0;    fpRate = 1.0
        for j in range(nc):
            if j != r:
                fpRateNum += confusionMat[j, r];        fpRateDen += confusionMat[j,:].sum()
        if fpRateDen != 0: fpRate = fpRateNum / fpRateDen
        print(f"Class ({r + 1}): {fScore:7.3} | {precision:9.3} | {recall:9.3}     | {fpRate:7.3}")
    print(f"\nAccuracy: {accuracy*100}%")
```

Reconocimiento de imágenes

- Programa base: **e6.py** (módulo *machine learning*)
  - **Métodos cv.ml.SVM**: Ajuste automatic de parámetros (C, Gamma)

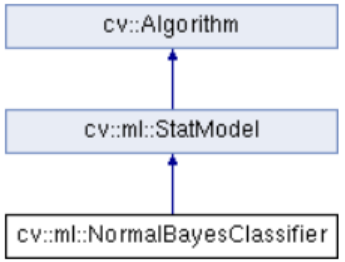
    - cv.ml.SVM.**trainAuto**(**samples**, **layout**, **responses** [, kFold [, Cgrid[, gammaGrid[, pGrid[, nuGrid [, coeffGrid[, degreeGrid[, balanced]]]]]]]]) → retval (bool)

      - **samples:** The input samples, floating-point matrix (np.float32 / cv.CV_32F)
      - **layout:** cv.ml.**ROW_SAMPLE** , cv.ml.**COL_SAMPLE**
      - **responses:** vector of responses associated with the training samples

      - **kFold**: (10) Cross-validation parameter. The training set is divided into kFold subsets. One subset is used to test the model, the others form the train set.
      - **Cgrid**: grid for C
      - **gammaGrid**: grid for gamma
      - **pGrid**: grid for p
      - **nuGrid**: grid for nu
      - **coeffGrid**: grid for coeff
      - **degreeGrid**: grid for degree
      - **balanced**: (False) If true and the problem is 2-class classification.

```
# Train SVM with auto tunning of SVM params: C, Gamma
retval = svm.trainAuto(trainData, cv.ml.ROW_SAMPLE, trainLabels, 5)       # kFold=5

print(f"SVM - C: {svm.getC()} Gamma: {svm.getGamma()}")
```

Reconocimiento de imágenes



- Otros clasificadores: (módulo *machine learning*)
  - Clase: **cv.ml_NormalBayesClassifier**
    - cv.ml.**NormalBayesClassifier_create**() → bayes (cv.ml_NormalBayesClassifier)
  - Métodos NormalBayes:
    - cv.ml_NormalBayesClassifier.**predictProb**(**inputs** [, outputs[, outputProbs[, flags]]]) → retval, outputs, outputProbs

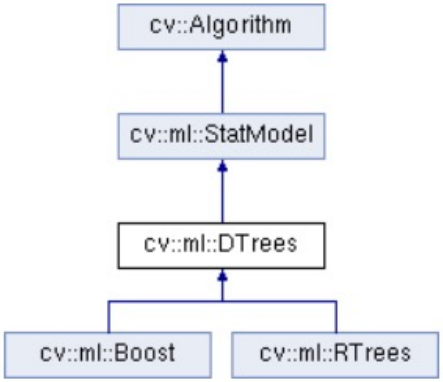
      outputs: output matrix of results. ndarray(m,1) (optional for single sample)

      outputProbs contains the output probabilities corresponding to each classs of result. **ndarray(m,nc)**

      retval: (float) prediction for single sample
    - cv.ml_StatModel.**train**(**samples**, **layout**, **responses**) → retval (bool)
    - cv.ml_StatModel.**train**(**trainData**[, **flags**]) → retval (bool)
    - cv.ml_StatModel.**predict**(**samples** [, **results** [, **flags**]]) → retval, results

```
bayes = cv.ml.NormalBayesClassifier_create()
bayes.train(trainData, cv.ml.ROW_SAMPLE, trainLabels)

# Test Normal Bayes (Prediction)
#retval, prediction = bayes.predict(testData)
retval, prediction, prob = bayes.predictProb(testData)

print(f"Probability: [", end='')
for r in range(len(prediction)):
    print(f"{prob[r, prediction[r]-1]:.3}, ", end='')
```

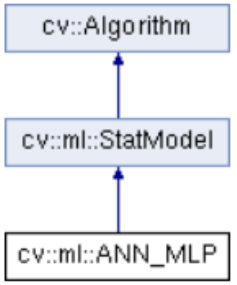
Reconocimiento de imágenes



- Otros clasificadores: (módulo *machine learning*)
  - Clase: **cv.ml_DTrees** (Decision Tree)
    - cv.ml.**DTrees_create**() → dtree (cv.ml_DTrees)
  - Métodos Dtrees:
    - cv.ml_DTrees.**setCVFolds**(**val**) val: (int) # prune a tree with K-fold cross-validation
    - cv.ml_DTrees.**setMaxDepth**(**val**). val: (int) # Max depth of the decision tree
    - cv.ml_DTrees.**setTruncatePrunedTree**(**val**) val: (bool) # If true then a pruning will be harsher
    - cv.ml_DTrees.**setUse1SERule**(**val**) val: (bool) # If true pruned branches are removed from the tree.
    - cv.ml_DTrees.**setUseSurrogates**(**val**) val: (bool) # If true then surrogate splits will be built
    - cv.ml_StatModel.**train**(**samples**, **layout**, **responses**) → retval (bool)
    - cv.ml_StatModel.**train**(**trainData**[, **flags**]) → retval (bool)
    - cv.ml_StatModel.**predict**(**samples** [, **results** [, **flags**]]) → retval, results

```
dtree = cv.ml.DTrees_create()

# Set up DTree's parameters
dtree.setCVFolds(0)             #  If cv_folds > 1 then prune a tree with K-fold cross-validation
dtree.setMaxDepth(10)           # Max depth of the decision tree

dtree.train(trainData, cv.ml.ROW_SAMPLE, trainLabels)
retval, prediction = dtree.predict(testData)
```

Reconocimiento de imágenes



- Otros clasificadores: (módulo *machine learning*)
  - Clase: **cv.ml_ANN_MLP** Neural Network – Perceptron Multicapa
    - cv.ml.**ANN_MLP_create**() → mlp (cv.ml_ANN_MLP)

  - Métodos Dtrees:
    - cv.ml_ANN_MLP.**setActivationFunction**(**type** [, param1[, param2]])
      - **type**: cv.ml.ANN_MLP_IDENTITY | ANN_MLP_SIGMOID_SYM | …
      - **param1:** The first parameter of the activation function, α. Default value is 0.
      - **param2:** The second parameter of the activation function, β. Default value is 0.

IDENTITY	Identity function: $f(x) = x$
SIGMOID_SYM	Symmetrical sigmoid: $f(x) = \beta * (1 - e^{-\alpha x})/(1 + e^{-\alpha x})$
GAUSSIAN	Gaussian function: $f(x) = \beta e^{-\alpha x * x}$
RELU	ReLU function: $f(x) = max(0, x)$
LEAKYRELU	Leaky ReLU function: for x>0 $f(x) = x$ and x<=0 $f(x) = \alpha x$

    - cv.ml_ANN_MLP.**setTrainMethod**(**method** [, param1[, param2]])
      # Training methods: cv.ml.ANN_MLP_BACKPROP | ANN_MLP_RPROP | ANN_MLP_ANNEAL
    - cv.ml_ANN_MLP.**setLayerSizes**(layer_sizes) DEBE CONFIGURARSE LA PRIMERA
      # integer **ndarray** specifying the number of neurons in each layer including the input and output layers
    - cv.ml_ ANN_MLP.**setTermCriteria**(**val**)

Reconocimiento de imágenes

- Otros clasificadores: (módulo *machine learning*)
  - Clase: **cv.ml_ANN_MLP.** Neural Network – Perceptron Multicapa
    - cv.ml.**ANN_MLP_create**() → mlp (cv.ml_ANN_MLP)

```
mlp = cv.ml.ANN_MLP_create()

# Set up MLP parameters
# FIRST layerSizes: integer vector with the number of neurons in each layer including the input and output layers.
mlp.setLayerSizes(np.array([trainData.shape[1], 5, len(labelRange)]))      # 1 hidden layer  with 5 neurons

# Activation Function: cv.ml.ANN_MLP_IDENTITY | ANN_MLP_SIGMOID_SYM |
# ANN_MLP_GAUSSIAN  | ANN_MLP_RELU  | ANN_MLP_LEAKYRELU
# SIGMOID: param1: alpha, param2: beta
mlp.setActivationFunction(cv.ml.ANN_MLP_SIGMOID_SYM, , param1=0.1, param2=1.5)

# Training methods: cv.ml.ANN_MLP_BACKPROP |  ANN_MLP_RPROP  | ANN_MLP_ANNEAL
mlp.setTrainMethod(cv.ml.ANN_MLP_BACKPROP)


mlp.setTermCriteria((cv.TermCriteria_MAX_ITER | cv.TermCriteria_EPS, 10000, 1e-6))
```

Reconocimiento de imágenes

- Otros clasificadores: (módulo *machine learning*)
  - Clase: **cv.ml_ANN_MLP.** Neural Network – Perceptron Multicapa
    - cv.ml.**ANN_MLP_create**() → mlp (cv.ml_ANN_MLP)
  - Métodos:
    - cv.ml_StatModel.**train**(**samples**, **layout**, **responses**) → retval (bool)
    - cv.ml_StatModel.**train**(**trainData**[, **flags**]) → retval (bool)
    - cv.ml_StatModel.**predict**(**samples** [, **results** [, **flags**]]) → retval, results

    - **responses** debe ser una matriz float32 con una columna por cada clase. En clasificación pondremos a 1 la columna correspondiente a la clase de la muestra y 0 las demás

```
# adapt trainLabels vector to a float32 matrix with one colum per class
trainLabelsMat = np.zeros((len(trainLabels), len(labelRange)), dtype=np.float32)
for row, val in enumerate(trainLabels):
        trainLabelsMat[row, val-1] = 1.0

mlp.train(trainData, cv.ml.ROW_SAMPLE, trainLabelsMat)
mlp.save("MLP.json")    # Save trained classifier

# Test MLP Neural Network (Prediction)
retval, predictionMat = mlp.predict(testData)

# convert predictionMat to vector: search for max response across classes (colums)
prediction = predictionMat.argmax(axis=1) + 1
```

Reconocimiento de imágenes

- Otros clasificadores: (módulo *machine learning*)
  - Clase: **cv.ml_Boost** Adaboost
    - cv.ml.**Boost_create**() → classifier (cv.ml_Boost)
  - Clase: **cv.ml_EM** Expectation/Maximization
    - cv.ml.**EM_create**() → classifier (cv.ml_EM)
  - Clase: **cv.ml_KNearest** Vecino más cercano
    - cv.ml.**KNearest_create**() → classifier (cv.ml_KNearest)
  - Clase: **cv.ml_LogisticRegression** Adaboost
    - cv.ml.**LogisticRegression_create**() → classifier (cv.ml_LogisticRegression)
  - Clase: **cv.ml_RTrees** Multiple Random Trees
    - cv.ml.**RTrees_create**() → classifier (cv.ml_RTrees)
  - Clase: **cv.ml_SVMSGD** Stochastic Gradient Descent SVM
    - cv.ml.**SVMSGD_create**() → classifier (cv.ml_SVMSGD)

Reconocimiento de imágenes

- Paquete *sckit-learn*
  - https://scikit-learn.org/stable/user_guide.html

Supervised Learning Estimators

Linear Regression
```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```
Support Vector Machines (SVM)
```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```
Naive Bayes
```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```
KNN
```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

Unsupervised Learning Estimators

Principal Component Analysis (PCA)
```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```
K Means
```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning	
`>>> lr.fit(X, y)`	Fit the model to the data
`>>> knn.fit(X_train, y_train)`	
`>>> svc.fit(X_train, y_train)`	
Unsupervised Learning	
`>>> k_means.fit(X_train)`	Fit the model to the data
`>>> pca_model = pca.fit_transform(X_train)`	Fit to data, then transform it

Prediction

Supervised Estimators	
`>>> y_pred = svc.predict(np.random.random((2,5)))`	Predict labels
`>>> y_pred = lr.predict(X_test)`	Predict labels
`>>> y_pred = knn.predict_proba(X_test)`	Estimate probability of a label
Unsupervised Estimators	
`>>> y_pred = k_means.predict(X_test)`	Predict labels in clustering algos

```
sklearn.gaussian_process: Gaussian Processes . . . . . .
sklearn.isotonic: Isotonic regression . . . . . .
sklearn.impute: Impute . . . . . .
sklearn.kernel_approximation Kernel Approximation . . .
sklearn.kernel_ridge Kernel Ridge Regression . . . . . .
sklearn.linear_model: Generalized Linear Models . . . . . .
sklearn.manifold: Manifold Learning . . . . . .
sklearn.metrics: Metrics . . . . . .
sklearn.mixture: Gaussian Mixture Models . . . . . .
sklearn.model_selection: Model Selection . . . . . .
sklearn.multiclass: Multiclass and multilabel classification . .
sklearn.multioutput: Multioutput regression and classification
sklearn.naive_bayes: Naive Bayes . . . . . .
sklearn.neighbors: Nearest Neighbors . . . . . .
sklearn.neural_network: Neural network models . . . . . .
sklearn.pipeline: Pipeline . . . . . .
sklearn.inspection: inspection . . . . . .
sklearn.preprocessing: Preprocessing and Normalization . . .
sklearn.random_projection: Random projection . . . . . .
sklearn.semi_supervised Semi-Supervised Learning . . . . .
sklearn.svm: Support Vector Machines . . . . . .
sklearn.tree: Decision Trees . . . . . .
sklearn.utils: Utilities . . . . . .
```

Classification Metrics

Accuracy Score	
`>>> knn.score(X_test, y_test)`	Estimator score method
`>>> from sklearn.metrics import accuracy_score`	Metric scoring functions
`>>> accuracy_score(y_test, y_pred)`	
Classification Report	
`>>> from sklearn.metrics import classification_report`	Precision, recall, f1-score
`>>> print(classification_report(y_test, y_pred))`	and support
Confusion Matrix	
`>>> from sklearn.metrics import confusion_matrix`	
`>>> print(confusion_matrix(y_test, y_pred))`	

RECONOCIMIENTO 6b

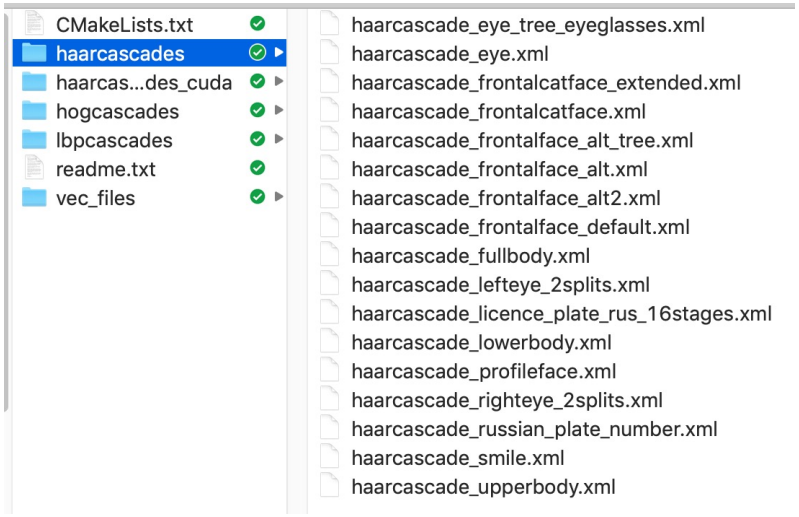
Módulo Object Detection (e6b.py)

- Clasificadores en cascada
- Detector de caras Harr
- Detector de peatones HOG

https://docs.opencv.org/4.x/d5/d54/group__objdetect.html

Reconocimiento de imágenes

- Clasificadores en Cascada: e6b.py (módulo object detection)
  Detectar caras: (Haar//LBP → cv.CascadaClassifier)
  - Clase: **cv.CascadeClassifier**
    - cv.**CascadeClassifier**( ) → classifier <CascadeClassifier object>
    - cv.**CascadeClassifier**(filename) → classifier <CascadeClassifier object>
      Carga datos del clasificador entrenado
  - Métodos:
    - cv.CascadeClassifier.**load**(filename) → bool
      Loads a classifier from a file
    - cv.CascadeClassifier.**empty**() → bool
      Checks whether the classifier has been loaded



- cv.CascadeClassifier.**detectMultiScale**(**image** [, scaleFactor [, minNeighbors [, flags [, minSize [, maxSize]]]]]) → objects

objects: tuple (x, y, width, height) rectangle contains the detected object

Reconocimiento de imágenes

- Clasificadores en Cascada: e6b.py (partimos del código de e1.py)

```
CASCADE_FACE_FILE = '../data/haarcascades/haarcascade_frontalface_alt.xml'
CASCADE_EYES_FILE = '../data/haarcascades/haarcascade_eye_tree_eyeglasses.xml'

# Load cascade trained classifiers
face_cascade = cv.CascadeClassifier(CASCADE_FACE_FILE)
eyes_cascade = cv.CascadeClassifier(CASCADE_EYES_FILE)

if face_cascade.empty() or eyes_cascade.empty():
    print(f"Error loading Cascade Classifiers")
    exit(-1)
```

```
gray_image = cv.cvtColor(capture, cv.COLOR_BGR2GRAY)
gray_image = cv.equalizeHist(gray_image)  # Normalize gray levels
# Detect faces
    faces = face_cascade.detectMultiScale(gray_image)
    for (x,y,w,h) in faces:
        center = (x + w // 2, y + h // 2)
        cv.ellipse(capture, center, (w // 2, h // 2), angle=0, startAngle=0, endAngle=360,
                                                color=(255, 0, 255), thickness=4)
        faceROI = gray_image[y:y + h, x:x + w]

        # -- In each face, detect eyes
        eyes = eyes_cascade.detectMultiScale(faceROI)
        for (x2, y2, w2, h2) in eyes:
            eye_center = (x + x2 + w2 // 2, y + y2 + h2 // 2)
            radius = int(round((w2 + h2) * 0.25))
            cv.circle(capture, eye_center, radius, color=(255, 0, 0), thickness=4)
```

Reconocimiento de imágenes

- Clasificadores en Cascada: e6b.py (módulo object detection)
  Detectar peatones: (HOG → cv.HOGDescriptor) (SVM classifier)
  - Clase: **cv.HOGDescriptor**
    - cv.**HOGDescriptor**( ) → classifier <HOGDescriptor object>
    - cv.**HOGDescriptor**(filename) → classifier <HOGDescriptor object>
      Carga datos del clasificador entrenado
  - Métodos:
    - cv.HOGDescriptor.**load**(filename) → bool
      Loads a classifier from a file
    - cv.HOGDescriptor.**setSVMDetector** (svmdetector)
      svmdetector: cv.HOGDescriptor.**getDefaultPeopleDetector**()
      cv.HOGDescriptor.**getDaimlerPeopleDetector**()
    - cv.HOGDescriptor.**detectMultiScale**(**image** [, hitThreshold [, winStride [, padding
      [, scale [, finalThreshold [, useMenshiftGrouping]]]]]) → foundLocations, foundWeigths

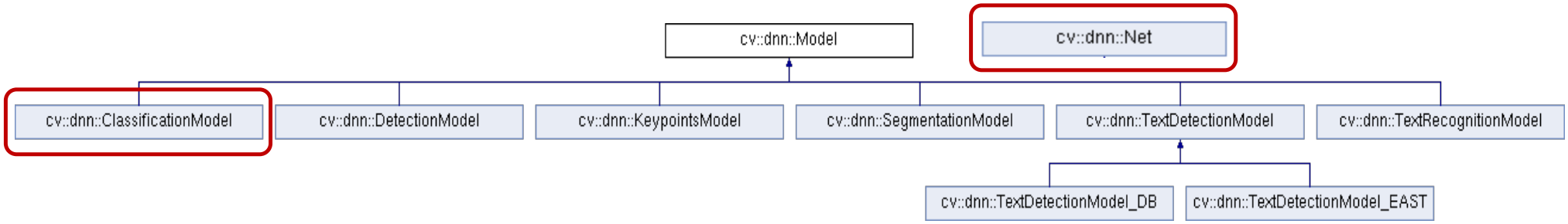
      foundLocations: list of tuples (x, y, w, h) rectangle contains the detected object
      foundWeigths: list that will contain confidence values for each detected object

Reconocimiento de imágenes

- Clasificadores en Cascada: e6b: HOG

```
# Configure pedestrian classifier
pedestrian_detector = cv.HOGDescriptor()
pedestrian_detector.setSVMDetector(cv.HOGDescriptor.getDefaultPeopleDetector())
```

```
# Detect pedestrians
    foundLocations, foundWeigths = pedestrian_detector.detectMultiScale(gray_image)

    for id in range(len(foundLocations)):
        (x,y,w,h) = tuple(foundLocations[id])

        cv.rectangle(capture, (x,y), (x+w, y+h), color=(0, 0, 200), thickness=2)
        cv.putText(capture, f"{foundWeigths[id]:.2}", (x+5,y-5), cv.FONT_HERSHEY_DUPLEX, 0.3,
                                        (0, 200, 200), 1, cv.LINE_AA)
```



REDES NEURONALES CONVOLUCIONALES (CNN)

Módulo Deep Learning (dnn) (e6c.py)

- Importar redes pre-entrenadas (Caffe/Darknet)
- Clasificación: AlexNet, GoogLeNet
- Detección/Clasificación: YOLO
- Transfer Learning (keras)
- Librerías: **opencv** / **keras-tensorflow-pytorch**

https://docs.opencv.org/4.5.5/d6/d0f/group__dnn.html

https://keras.io/api/
https://www.tensorflow.org/api_docs/python/tf/all_symbols

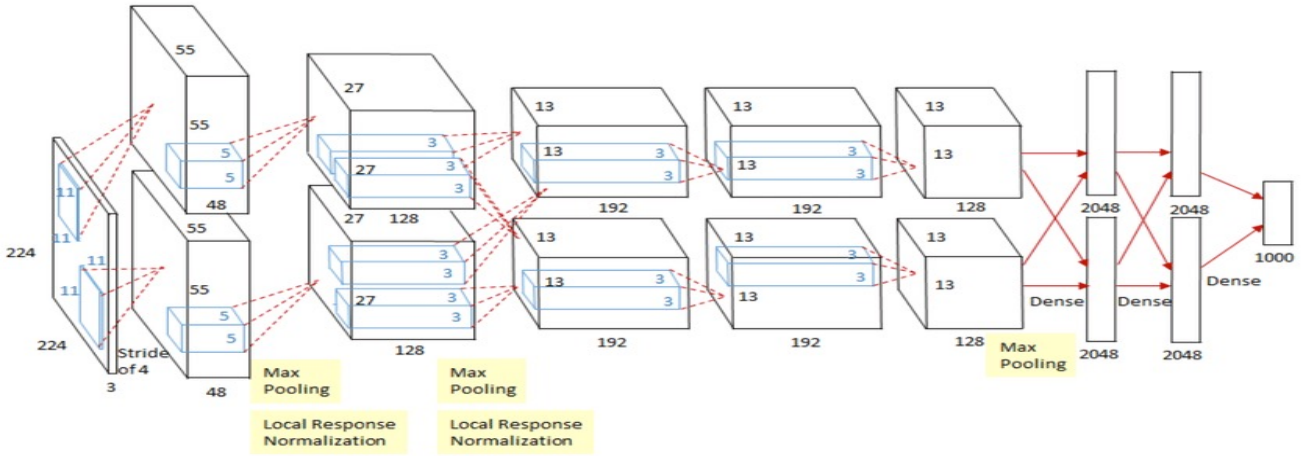
Deep Learning (CNN)

- Redes CNN pre-entrenadas Clasificación, formatos (frameworks):
  - Cafee, Tensorflow, PyTorch, Darknet, ONNX
  - Modelos Caffe: fichero **CaffeModels.zip** (http://umh1782.edu.umh.es/python/)
    - **GoogLeNet**: (1000 clases ImageNet dataset) (input: 224x224)
      - https://arxiv.org/pdf/1409.4842.pdf
    - **AlexNet** (ImageNet): (1000 clases ImageNet dataset) (input: 227x227)
      - https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf (Alex Krizhevsky)
    - **CaffeNet**-ImageNet : variación de AlexNet (1000 clases ImageNet dataset)
    - **Fine-Tunning**: con dataset Flickr Style (20 clases PASCAL-VOC-12)
    - **R-CNN**: (200 clases ILSVRC13). (input: 227x227)
      - https://arxiv.org/pdf/1407.3867.pdf
  - Datasets:
    - **ImageNet** Large Scale Visual Recognition Challenge (ILSVRC):
      - https://image-net.org/challenges/LSVRC/
    - The **PASCAL** Visual Object Classes (PASCAL-VOC):
      - http://host.robots.ox.ac.uk/pascal/VOC/

Deep Learning (CNN)

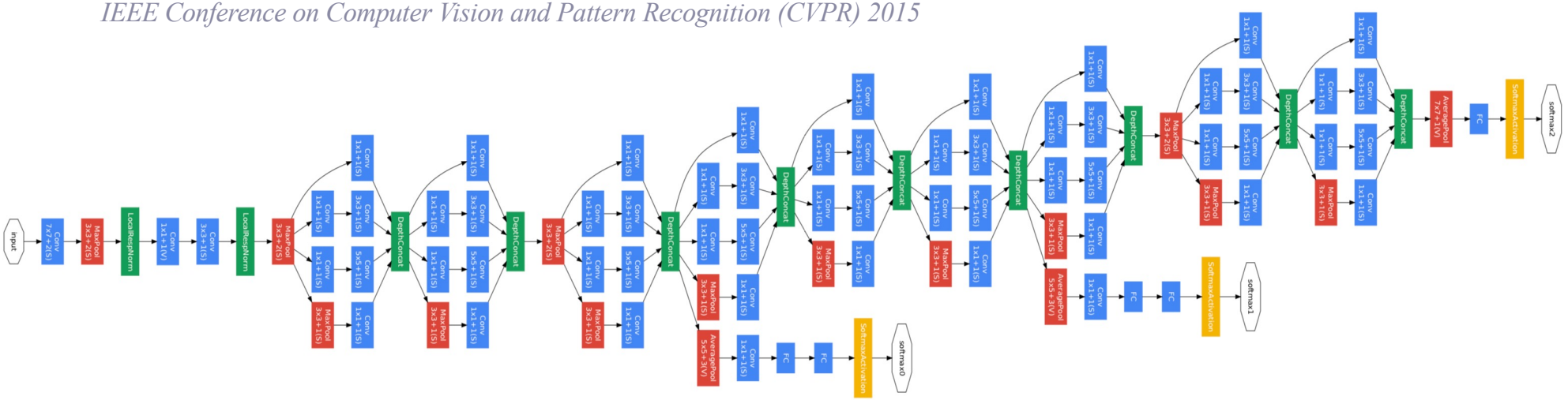
- **AlexNet:** (8/24 capas - 60Millones parámetros) - input 227x227x3

"ImageNet Classification with Deep Convolutional Neural Networks" Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton Communications of the ACM Volume 60 Issue 6 June 2017



- **GoogleNet:** (22/144 capas-12Millones parámetros) - input 224x224x3

"Going Deeper with Convolutions" C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2015



Reconocimiento de imágenes

- Programa: **e6c.py** (módulo Deep Learning)
  - Importar redes pre-entrenadas (Caffe)
- Clases: cv.dnn_Model, cv.dnn_ClassificationModel (OPCIÓN A)
  - cv.**dnn_Model**(network) → <dnn_Model object>
  - cv.**dnn_Model**(model [, config]) → <dnn_Model object>
  - cv.**dnn_ClassificationModel**(network) → <dnn_ClassificationModel object>
  - cv.**dnn_ClassificationModel**(model [, config]) → <dnn_ClassificationModel object>

    model: (str) Nombre fichero binario con los pesos de la red entrenados
    *.caffemodel (**Caffe**), *.pb (**TensorFlow**), *.t7 | *.net (**Torch**), *.weights (**Darknet**)
    *.bin (**DLDT**), *.onnx (**ONNX**)

    config: (str) Nombre fichero de texto con la configuración de la red
    *.prototxt (**Caffe**), *.pbtxt (**TensorFlow**), *.cfg (**Darknet**), *.xml (**DLDT**)
- Métodos:
  - cv.dnn_Model.**setInputParams**([, **scale** [, **size** [, **mean** [, **swapRB** [, **crop**]]]]])
    blob(N, c, h, w) = scale * resize(frame(y, x, c)) - mean(c)) (Ver formato en crear blob dnn_Net)
  - cv.dnn_Model.**predict**(**frame** [, outs]) → outs

    frame: Crea el blob de entrada, propaga (ejecuta) la red y devuelve los blobs de salida.

    outs tuple(N) de ndarray(1,nc) blobs, almacena los resultados de los nodos de la capa de salida.
    (confianza - activación nodo)
  - cv.dnn_ClassificationModel.**classify**(**frame**) → classId, conf

    frame: Crea el blob de entrada, propaga (ejecuta) la red y devuelve la mejor predicción (máximo nodo)
    classId: (int)
    conf: (float) valor de probabilidad

Reconocimiento de imágenes

- Programa: **e6c.py** (módulo Deep Learning)
  - Importar redes pre-entrenadas (Caffe)
- Clase: cv.dnn_Net (OPCIÓN B)
  - cv.dnn.**readNet**(**model** [, **config** [, **framework**]]) → <dnn_Net object>

    model: (str) Nombre fichero binario con los pesos de la red entrenados:
    *.caffemodel (**Caffe**), *.pb (**TensorFlow**), *.t7 | *.net (**Torch**), *.weights (**Darknet**)
    *.bin (**DLDT**), *.onnx (**ONNX**)

    config: (str) Nombre fichero de texto con la configuración de la red:
    *.prototxt (**Caffe**), *.pbtxt (**TensorFlow**), *.cfg (**Darknet**), *.xml (**DLDT**)

    framework: (str) Opcional, etiqueta con el nombre del Framework para determinar el formato
- Métodos:
  - cv.dnn_Net.**setInput**(**blob** [, name [, scalefactor[, mean]]])

    blob: ndarray(N,c,h,w) . Nuevo blob. Debe tener profundidad: cv.**CV_32**F ó cv.**CV_8U**

    name: (str) Nombre opcional de la capa de entrada. Una imagen: N=1

    scalefactor: (float) Opcional, escala de normalización.

    mean: (tuple) Opcional media a restar a los valores de imagen (b,g,r) ó (r,g,b) si swapRB es True

    blob(n, c, y, x) = scale * resize(frame(y, x, c)) - mean(c))
  - cv.dnn_Net.**forward**( ) → outputBlobs tuple(N) de ndarray(1,nc) (confianza)
  - cv.dnn_Net.**empty**( ) → bool
  - cv.dnn_Net.**dump**( ) → retval (str). Crea cadena con la descripción de la red
  - cv.dnn_Net.**setPreferableBackend**(backendId) Para elegir aceleración Hardware
  - cv.dnn_Net.**setPreferableTarget**(targetId) También disponibles para la clase cv.dnn_Model

Reconocimiento de imágenes

- Programa: **e6c.py** (módulo Deep Learning)
- Clase: cv.dnn_Net (OPCIÓN B)
- Funciones de formateo entradas de la red:
  - cv.dnn.**blobFromImage**(**image** [, scalefactor[, size[, mean[, swapRB[, crop[, ddepth]]]]]]) → blob
  - cv.dnn.**blobFromImages**(**images**[, scalefactor[, size[, mean[, swapRB[, crop[, ddepth]]]]]]) → blob
  - cv.dnn.**imagesFromBlob**(**blob** [, images]) → images

blob: ndarray(N,c,h,w) 4-dimensional blob (N: número de imágenes)

image: (ndarray) imagen de entrada (1-, 3- or 4-channels).

scalefactor: (float) multiplicador para los valores de la imagen. **(def: 1.0)**

size: (tuple) dimensiones imagen del blob (w,h). Depende la la capa de entrada de la red **ImageNet**: (224,224)

mean: (tuple) media a restar a los valores de imagen (b,g,r) ó (r,g,b) si swapRB está activado Hace que la respuesta de la red sea invariante a la iluminación.

swapRB: (bool) flag, indica que se deben permutar el primer y tercer canal en imágenes de 3 canales (b,g,r) → (r,g,b). **(def: false)**

crop: (bool) flag, indica si la imagen debe ser recortada después de redimensionarla Preserva relación alto/ancho de la imagen. (**def: false).**

ddepth: Profundidad (tipo) del blob de salida. cv.**CV_32F (def.)** ó cv.**CV_8U**.

blob(n, c, y, x) = scale * resize(frame(y, x, c)) - mean(c))

Reconocimiento de imágenes

- Programa: **e6c.py** (módulo Deep Learning)
- Clase: cv.dnn_Net (OPCIÓN B)
- Funciones de gestión de capas:
  - cv.dnn_Net.**getLayerNames**( ) → retval list(str) lista de strings: nombre de capas
  - cv.dnn_Net.**getLayerId**(layer) → retval int nombre capa a índice (int). (*empieza en 1*)
  - cv.dnn_Net.**getUnconnectedOutLayers**( ) → retval list(int): índices capas salida (*empieza en 1*)
  - cv.dnn_Net.**getUnconnectedOutLayersNames**( ) → retval list(str): nombre capas salida

  - cv.dnn_Net.**getLayerTypes**( ) → retval list(str): nombre tipos de capa
  - cv.dnn_Net.**getLayersCount**(layerType) → retval int numero capas por tipo (str)

- Funciones estimación tiempo de computo:
  - cv.**TickMeter**() → <TickMeter object> Medida de tiempos de computo y FPS
  - cv.TickMeter.**start**( ) starts counting ticks.
  - cv.TickMeter.**stop**( ) stop counting ticks.
  - cv.TickMeter.**reset**( ) resets internal values.
  - cv.TickMeter.**getFPS**( ) → retval
  - cv.TickMeter.**getTimeMilli**( ) → retval

Reconocimiento de imágenes

- Programa: **e6c.py** (módulo Deep Learning) (partiremos de **e1.py**)
- Importar redes pre-entrenadas (Caffe)

```
MODEL_FILE = '../caffe/bvlc_alexnet/bvlc_alexnet.caffemodel'
CONFIG_FILE = '../caffe/bvlc_alexnet/deploy.prototxt'
LABELS_FILE = '../caffe/bvlc_alexnet/imagenet-labels.txt'
BLOB_SIZE = (227, 227)      # input layer size

tm = cv.TickMeter()                 # TickMeter object to calculate FPS

# Load CNN data
network = cv.dnn.readNet(model=MODEL_FILE, config=CONFIG_FILE, framework='Caffe')
networkModel = cv.dnn_ClassificationModel(network)

# Load labels
with open(LABELS_FILE, 'r') as file:
      labels = file.read().rstrip('\n').split('\n')

# Show network model data
print(network.dump())

print(f"#Classes: {len(labels)}")
print(f"#Layers: {len(network.getLayerNames())}")
print(f"#OutputLayers: {len(network.getUnconnectedOutLayers())} - ", end='')
print(network.getUnconnectedOutLayersNames() , network.getUnconnectedOutLayers())
```

Reconocimiento de imágenes

- Programa: **e6c.py** (módulo Deep Learning) (partiremos de **e1.py**)
- Usando la clase: cv.dnn_ClassificationModel (OPCIÓN A)

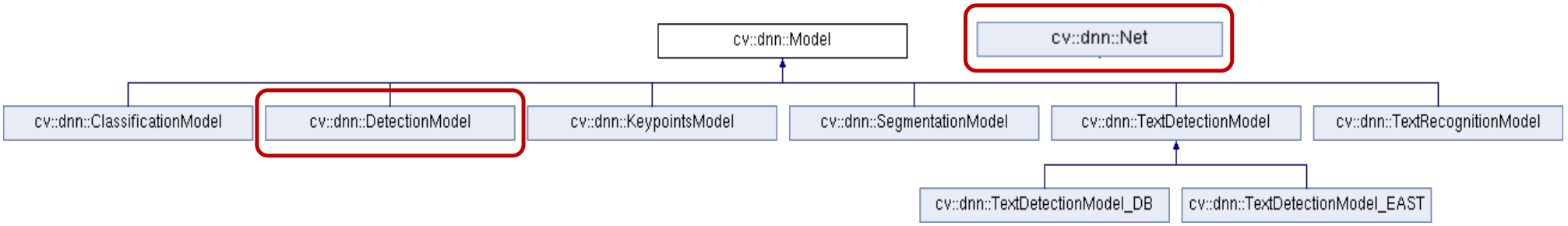
```
……..

tm.start()  # start processing cycle

#  image processing code using  dnn_ClassificationModel
networkModel.setInputParams(scale=1.0, size=BLOB_SIZE,
                                        mean=capture.mean(axis=(0, 1)) , swapRB=True,  crop=True)

classId, conf = networkModel.classify(capture)


tm.stop()  # end processing cycle
# Show FPS on image
cv.putText(capture, f"FPS: {tm.getFPS():.1f} ", org=(3, 20), fontFace=cv.FONT_HERSHEY_DUPLEX,
                            fontScale=0.5, color=(0, 0, 255), thickness=1, lineType=cv.LINE_AA)

if conf > 0.3:
    cv.putText(capture, f"({classId}) {conf*100:5.2f}% - {labels[classId]} ", (90, 20),
                        cv.FONT_HERSHEY_DUPLEX, 0.5, (0, 200, 0), thickness=1, lineType=cv.LINE_AA)

……
```

Reconocimiento de imágenes

- Programa: **e6c.py** (módulo Deep Learning) (partiremos de **e1.py**)
- Usando la clase cv.dnn_Net (OPCIÓN B)

```
.....

# image processing code using dnn_Net
blob = cv.dnn.blobFromImage(capture, scalefactor=1.0, size=BLOB_SIZE,
                            mean=capture.mean(axis=(0, 1)), swapRB=True, crop=True)
network.setInput(blob)
outputBlobs = network.forward()

classId = np.argmax( outputBlobs[0] )
conf = np.max( outputBlobs[0] )

.....
```

- Probar el resto de redes pre-entrenadas de clasificación:
  - GoggleNet: (1000 clases) (224x224)
  - CaffeNet: (1000 clases) (227x227)
  - R-CNN: (200 clases) (227,227) - outputBlobs (tanh/sigmoid) $softmax, p(i) = \frac{e^{c(i)}}{\sum_j e^{c(j)}}$
  - Finetune-Flicker: (20 clases) (227x227)



REDES NEURONALES CONVOLUCIONALES (CNN)

Módulo Deep Learning (dnn) (e6d.py)

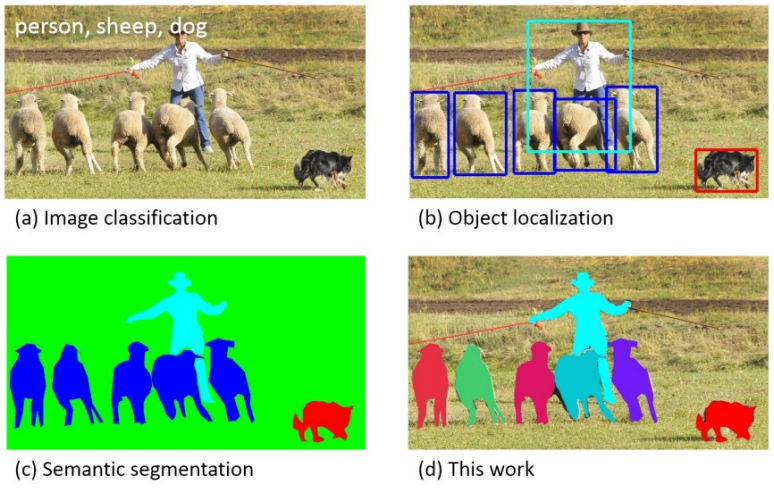
- Detección/Clasificación: YOLO

https://docs.opencv.org/4.5.5/d6/d0f/group__dnn.html

https://pjreddie.com/darknet/yolo/

Deep Learning (CNN)

- Redes CNN pre-entrenadas, Detección/Clasificación:
  - (http://umh1782.edu.umh.es/python/)
  - Modelos Darknet: fichero **DarknetModels.zip**
    - **YOLOv3**: (80 clases COCO dataset - segmentado) (input: 608x608) Detección de Ventana. Adaptable a blobs de menor resolución (320x320) (608x608) ↑ FPS
      https://pjreddie.com/darknet/yolo/ (Joseph Redmon, Ali Farhadi)
      https://arxiv.org/pdf/1804.02767.pdf
    - **YOLOv3-tiny** : 80 clases COCO dataset) - input 416x416x3 (Rápido – mayor error)
  - Modelos Tensorflow:
    - https://github.com/tensorflow/models/tree/master/community
  - Modelos ONNX:
    - https://github.com/onnx/models
  - Datasets:
    - **MS COCO** Common Objets in Context
      80 clases, etiquetado semántico de cada región
      - https://cocodataset.org

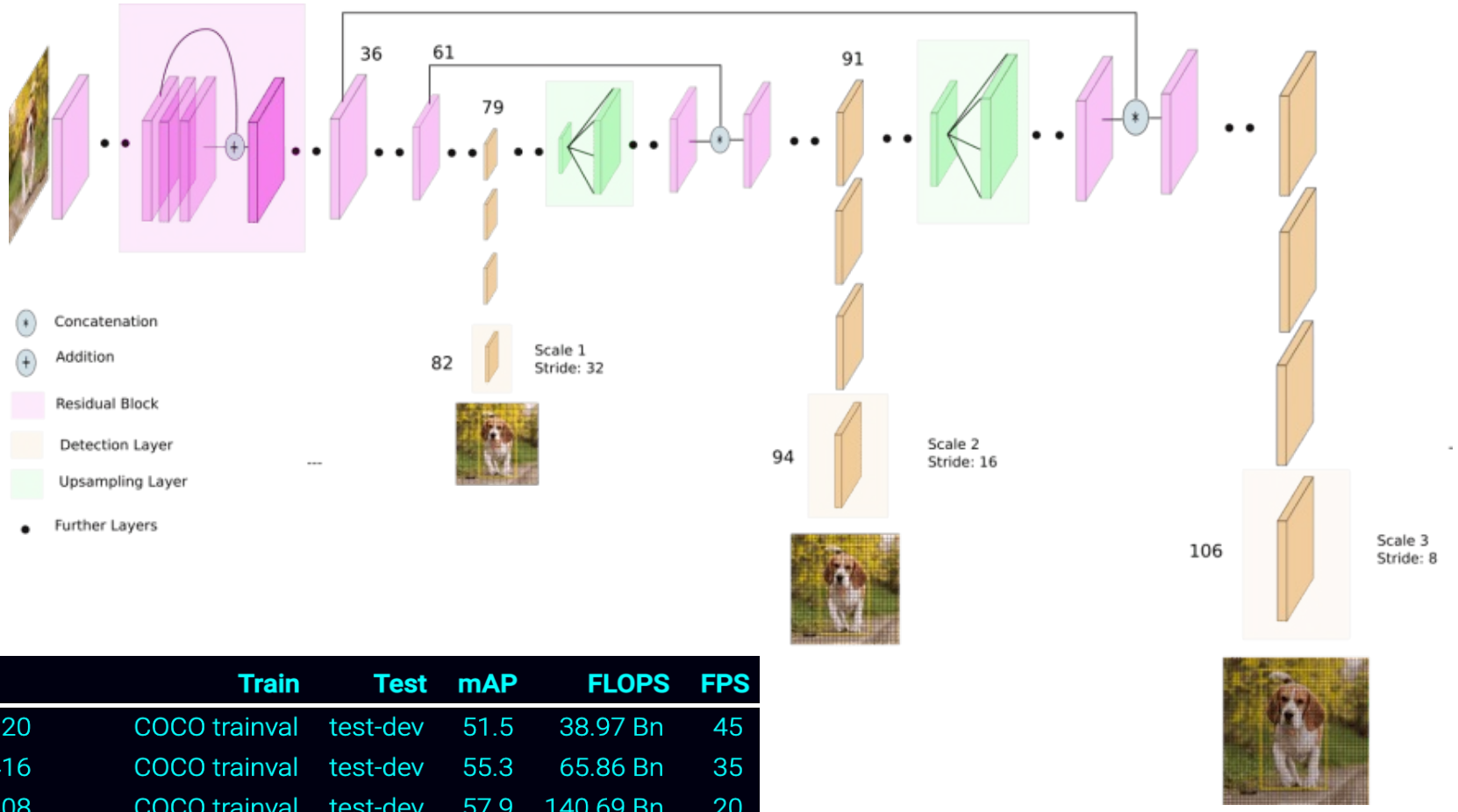


Reconocimiento de imágenes

- YOLOv3 : 106 capas - 60Millones parámetros) - input 608x608x3

"You Only Look Once: Unified, Real-Time Object Detection"
Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi CVPR (2016)

"YOLOv3: An Incremental Improvement"
Joseph Redmon, Ali Farhadi CVPR (2018)



Model	Train	Test	mAP	FLOPS	FPS
YOLOv3-320	COCO trainval	test-dev	51.5	38.97 Bn	45
YOLOv3-416	COCO trainval	test-dev	55.3	65.86 Bn	35
YOLOv3-608	COCO trainval	test-dev	57.9	140.69 Bn	20
YOLOv3-tiny	COCO trainval	test-dev	33.1	5.56 Bn	220
YOLOv3-spp	COCO trainval	test-dev	60.6	141.45 Bn	20

Reconocimiento de imágenes

- Programa: **e6d.py** (módulo Deep Learning)
  - Importar redes pre-entrenadas Darknet-YOLO

- Clases: cv.dnn_DetectionModel (YOLO9000 no funciona con las clases de OpenCV)
  - cv.**dnn_DetectionModel**(network) → <dnn_DetectionModel object>
  - cv.**dnn_DetectionModel**(model [, config]) → <dnn_DetectionModel object>

- Métodos:
  - cv.dnn_Model.**setInputParams**([, scale [, size [, mean [, swapRB [, crop]]]]])
    IMPORTANTE: YOLO está entrenada con blobs con valores de color entre [0-1]
    scale = 1.0/255

  - cv.dnn_DetectionModel.**detect**(**frame** [, confThreshold[, nmsThreshold]])
    → classIds, confidences, boxes
    frame: Crea el blob de entrada, propaga (ejecuta) la red y devuelve detecciones
    confThreshold (float) Umbral para filtrar ventana por confianza (Def. 0.5)
    nmsThreshold (float) Umbral usado en NMS (non maximum suppression) (Def. 0.0)

    classIds: ndarray(nd) (int)
    confidences: ndarray(nd) (float) valor de probabilidad
    boxes: ndarray(nd,4) Tuples (x,y,w,h). Conjunto de ventanas

  - cv.dnn_DetectionModel.**setNmsAcrossClasses**(value) → retval (value: bool)

Reconocimiento de imágenes

- Programa: **e6d.py** (módulo Deep Learning)
- Clase: cv.dnn_Net (OPCIÓN B)
- Métodos:
  - cv.dnn.**blobFromImage**(**image** [, scalefactor[, size[, mean[, swapRB[, crop[, ddepth]]]]]]) → blob

    IMPORTANTE: YOLO está entrenada con blobs con valores de color entre [0-1]

    scale = 1.0/255
  - cv.dnn_Net.**forward**( ) → outputBlobs. tuple (N) de ndarray(1,nc)

    Solo proporciona la respuesta de la última capa de salida de la red (YOLOv3: capa a mayor escala)

    network.**getUnconnectedOutLayersNames**()
  - cv.dnn_Net.**forward**(outputNames) → outputBlobs. **Tupla** de ndarray(1,nc)

    outBlobNames: nombre de las capas de salida para las que queremos procesar la red

    Permite extraer de forma separada la respuesta de la red a diferentes escalas (YOLO: 3 predictores)

5 nodos regresión | Num. Clases (80) nodos clasificación

outputBlobs[i]	cx	cy	w	h	conf	c1	c2	c3	…..	cn
	0.1	0.2	0.2	0.1	0.9	0	0	0.9	….	0
Detección ventana →	⇣	⇣	⇣	⇣	⇣	⇣	⇣	⇣	….	⇣

Box center *frame.width

  - Los valores de (cx,cy,w,h) están escalados entre [0-1]: multiplicar por la resolución de la imagen
  - cv.dnn.**NMSBoxes**(**bboxes**, **scores**, **score_threshold**, **nms_threshold**[, eta[, top_k]]) → indices

    Implementa filtro NMS (non maximum suppression) para las ventanas (bounding boxes) solapadas, con su correspondiente confianza (scores). Se queda con la de mayor probabilidad

Reconocimiento de imágenes

- Programa: **e6d.py** (módulo Deep Learning) (partiremos de **e6c.py**)
- Importar redes pre-entrenadas (Darknet-YOLO)

```
MODEL_FILE = '../darknet/yolov3.weights'
CONFIG_FILE = '../darknet/yolov3.cfg'
LABELS_FILE = '../darknet/coco.names'
BLOB_SIZE = (608, 608)      # native input layer size
BLOB_SIZE = (320, 320)      # downsized input layer size (increase FPS)

tm = cv.TickMeter()                 # TickMeter object to calculate FPS

# Load CNN data
network = cv.dnn.readNet(model=MODEL_FILE, config=CONFIG_FILE, framework='Darknet')
networkModel = cv.dnn_DetectionModel(network)

# Load labels
with open(LABELS_FILE, 'r') as file:
    labels = file.read().rstrip('\n').split('\n')

# create a list of random colors for each label
colors = np.random.randint(low=0, high=256, size=(len(labels),3))

# Open camera object
……
# Increase camera resolution
camera.set(cv.CAP_PROP_FRAME_WIDTH, 960)
camera.set(cv.CAP_PROP_FRAME_HEIGHT, 720)
```

Reconocimiento de imágenes

- Programa: **e6d.py** (módulo Deep Learning) (partiremos de **e6c.py**)
- Procesamiento clase dnn_ClassificationModel (Darknet-YOLO)

```
tm.start()  # start processing cycle
# Important: input blob color levels must be scaled to [0-1]
networkModel.setInputParams(scale=1.0/255, size=BLOB_SIZE, mean=capture.mean(axis=(0, 1)),
                                        swapRB=True)
networkModel.setNmsAcrossClasses(True)  # False: NMS only for bboxes of the same class

classIds, confidences, boxes = networkModel.detect( capture, confThreshold=0.3, nmsThreshold=0.2)

tm.stop()  # end processing cycle
# Show FPS on image
.....
```

```
# Show Detected Bounding Boxes
for idx, classId in enumerate(classIds):
    confidence = confidences[idx]
    (x, y, w, h) = boxes[idx]
    color = colors[classId]
    label = labels[classId]

    cv.rectangle(capture, (x, y), (x + w, y + h), color, thickness=2)
    text = f"{label}: {confidence*100:.1f}%"
    textSize, baseline = cv.getTextSize(text, cv.FONT_HERSHEY_DUPLEX, 0.3, 1)

    cv.rectangle(capture, (x, y), (x + 10 + textSize[0], y - 10 - textSize[1]), color, thickness=cv.FILLED)
    cv.putText(capture, text, (x + 5, y - 5), cv.FONT_HERSHEY_DUPLEX, 0.3, (255, 255, 255), 1, cv.LINE_AA)
```

Reconocimiento de imágenes

- Paquete keras https://keras.io/api/

Keras

Keras is a powerful and easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
                    activation='relu',
                    input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

Model Training

```
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               verbose=1,
               validation_data=(x_test4,y_test4))
```

Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x_test,
                            y_test,
                            batch_size=32)
```

Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Conv2D(64,(3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

Recurrent Neural Network (RNN)

```
>>> from keras.klayers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

Prediction

```
>>> model3.predict(x_test4, batch_size=32)
>>> model3.predict_classes(x_test4,batch_size=32)
```

Save/ Reload Models

```
>>> from keras.models import load_model
>>> model3.save('model_file.h5')
>>> my_model = load_model('my_model.h5')
```