RECONOCIMIENTO

Módulo Machine Learning (ej6.cpp)

- 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

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

https://docs.opencv.org/4.5.5/d3/dd5/tutorial_table_of_content_other.html

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

Reconocimiento de imágenes

- Programa base: **ej6.cpp**
- Leer datos entrenamiento Titere/Weka: módulo **titere.hpp**

  - bool titere::**readWekaData**(string **file**, string &**labelColum**, vector<string> &**attributes**,
    cv::Mat &**dataMat**, cv::Mat &**label**, vector<string> &**labelName**, vector<string> &**labelRange**)

    Read Weka arff data file
    inputs : **file**: (string) arff file
    input/output: **labelColum** : str column name for response(label) (last colum if "")
    attributes : (vector<string>) with selected column names (all if empty)
    output: **dataMat**: cv::Mat(m,n, CV_32F) rows: samples, cols features
    label: cv::Mat(m,1, CV_32F) index (1-reference) class for each sample
    labelName: vector<string> label for each sample
    input/output:
    labelRange : (vector<string>) with the range of labels (if empty, it is obtained from data)

  - void titere::**showPerformance**(vector<int> **label**, vector <int> **prediction**, vector<string> **labelRange**)

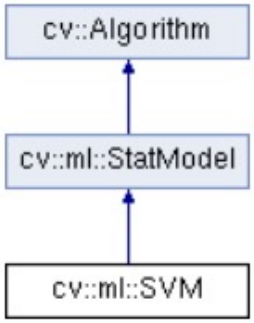
    Shows the classification accuracy and confusion matrix
    F-Score | Precision | Recall/TPRate | FPRate

Reconocimiento de imágenes

- Programa base: **ej6.cpp**

```
# include "titere.hpp"

string DATA_FILE = "train.arff";          // default data file
string TEST_FILE = "test.arff";          // default data file
string CLASSIFIER = "svm";               // svm | bayes | dtree | mlp

vector<string> attributes = { "Compacidad", "Excentricidad", "Rel_Invar_1", "Rel_Invar_2" };
string labelColum = "Pieza";
vector<string> labelRange;

cv::Mat trainData, trainLabels;
vector<string> trainLabelsName;
cv::Mat testData, testLabels;
vector<string> testLabelName;

titere::readWekaData(DATA_FILE, labelColum, attributes, trainData, trainLabels, trainLabelsName, labelRange);
titere::readWekaData(TEST_FILE, labelColum, attributes, testData, testLabels, testLabelName, labelRange);

// Display data
for (int i = 0; i < attributes.size(); i++)
        cout << attributes[i] << " ";
cout << labelColum << " (Num/Label)" << endl;

for (int i = 0; i < trainData.rows; i++)
        cout << trainData.row(i) << " " << trainLabels.row(i) << " " << trainLabelsName[i] << endl;

cout << "LabelRange: ";   for (int i = 0; i < labelRange.size(); i++)        cout << labelRange[i] << ",";
```

Reconocimiento de imágenes



- Programa base: **ej6.cpp** (módulo *machine learning*)
  - Clase: **cv::ml::SVM** (Support Vector Machine)
    - cv::ml::SVM::**create**() → svm (cv::**Ptr**<cv::ml::**SVM**>)

  - Métodos cv::ml::SVM:
    - cv::ml::SVM::**setType**(int **val**) cv::ml::**SVM_C_SVC** | **SVM_NU_SVC** | **SVM_ONE_CLASS** | **SVM_EPS_SVR** | **SVM_NU_SVR**
    - cv::ml::SVM::**setKernel**(int **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**(double **va**) POLY | RBF | SIGMOID | CHI2
- cv::ml::SVM::**setCoref0**(double **va**) POLY | SIGMOID
- cv::ml::SVM::**setDegree**(double **va**) POLY

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

- cv::ml::SVM::**getSupportVectors**() → retval cv::Mat(nsv, nfeatures, CV_32F)

Reconocimiento de imágenes

- Programa base: **ej6.cpp** (módulo *machine learning*)
  - Clase: **cv::ml::SVM** (Support Vector Machine)
    - cv::ml::SVM::**create**() → svm (cv::**Ptr**<cv::ml::**SVM**>)

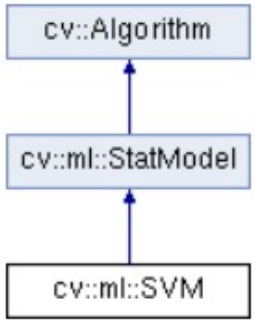
  - Métodos SVM:

    - cv::ml::SVM::**setTermCriteria**(const cv::**TermCriteria** & **val**)
      Finalización del algoritmo de optimización de entrenamiento

      val: cv::TermCriteria::**TermCriteria** (int **type**, int **maxCount**, double **epsilon**)

      type: 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: **ej6.cpp** (módulo *machine learning*)
  - Clase base (train/predict) : **cv::ml::StatModel**
    - cv::ml::StatModel::**train**(InputArray **samples**, int **layout**, InputArray **responses**) → retval (bool)
    - cv::ml::StatModel::**train**(Ptr<TrainData> &**trainData**, int **flags**=0) → retval (bool)
      - **samples:** The input samples, floating-point matrix cv::Mat(ns, nc, **CV_32F**)
      - **layout:** cv::ml::**ROW_SAMPLE** , cv::ml::**COL_SAMPLE**
      - **responses:** responses associated with the training samples cv::Mat(ns, nc, **CV_32S**) int32
      - **trainData:** training data created with cv::ml::TrainData::**create**
      - **retval**: (bool)
    - cv::ml::StatModel::**predict**(InputArray **samples**, OutputArray **results**, int **flags**=0) → retval (float)
      - **samples:** The input samples, floating-point matrix cv::Mat(ns, nc, **CV_32F**)
      - **results:** output matrix of results. cv::Mat(ns, 1, **CV_32F**) float32 (optional for single sample)
      - **retval**: (float) prediction for single sample
  - Clase cv::Algorithm
    - (Método) cv::Algorithm::**save**(string &filename) Guarda el modelo entrenado (YAML)
    - (Función) cv::ml::SVM ::**load**(string &filename) → cv::Ptr<cv::ml::SVM> lee el modelo entrenado
- **OpenCV**: conversion automática entre **cv::Mat** y **vector<int>**, **vector<float>**, **vector<double>**

Reconocimiento de imágenes

- Programa base: **ej6.cpp** **SVM** (módulo machine learning)

```
// create classifier
cv::Ptr<cv::ml::SVM> svm = cv::ml::SVM::create();

// Configure SVM Classifier
// SVM Type: cv::ml::SVM::C_SVC | NU_SVC | ONE_CLASS |  EPS_SVR  | NU_SVR
svm->setType(cv::ml::SVM::C_SVC); // C-Support Vector Classification

// Kernel type:  cv::ml::SVM::LINEAR | POLY | RBF | SIGMOID | CHI2  | INTER  | 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(cv::TermCriteria::MAX_ITER + cv::TermCriteria::EPS, 10000, 1e-6));

// Train SVM
bool retval = svm->train(trainData, cv::ml::ROW_SAMPLE, trainLabels);
cv::Mat sv = svm->getSupportVectors();              // get the support vectors
cout << "Support Vectors: " << sv.rows << endl;
svm->save("SVM.yaml");                  // Save trained classifier

// Test SVM(Prediction)
cv::Mat prediction;
svm->predict(testData, prediction);

cout << "Predicted: " << prediction.t() << endl;
cout << "Real:      " << testLabels.t() << endl;
titere::showPerformance(testLabels, prediction, labelRange)
```

Reconocimiento de imágenes

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

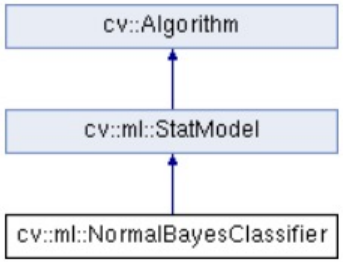
    - cv.ml.SVM.**trainAuto**(**samples**, **layout**, **responses** [, int kFold [, Cgrid[, gammaGrid[, pGrid[, nuGrid [, coeffGrid[, degreeGrid[, bool balanced]]]]]]]]) → retval (bool)

      - **samples:** The input samples, floating-point matrix cv::Mat(ns, nc, **CV_32F**)
      - **layout:** cv::ml::**ROW_SAMPLE** , cv::ml::**COL_SAMPLE**
      - **responses:** responses associated with the training samples cv::Mat(ns, nc, **CV_32S**) int32

      - **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. (cv::ParamGrid)
      - **gammaGrid**: grid for gamma. (cv::ParamGrid)
      - **pGrid**: grid for p. (cv::ParamGrid)
      - **nuGrid**: grid for nu. (cv::ParamGrid)
      - **coeffGrid**: grid for coeff. (cv::ParamGrid)
      - **degreeGrid**: grid for degree. (cv::ParamGrid)
      - **balanced**: (false) If true and the problem is 2-class classification.

```
// Train SVM with auto tunning of SVM params: C, Gamma
bool retval = svm->trainAuto(trainData, cv::ml::ROW_SAMPLE, trainLabels, 5);    // kFold=5

cout << "SVM - C: " << svm->getC() << " - Gamma: " << svm->getGamma() << endl;
```

Reconocimiento de imágenes



- Otros clasificadores: (módulo *machine learning*)
  - Clase: **cv::ml::NormalBayesClassifier**
    - cv::ml::NormalBayesClassifier::**create**() → bayes (cv::**Ptr**<cv::ml::**NormalBayesClassifier** >)

  - Métodos NormalBayes:
    - cv::ml::NormalBayesClassifier::**predictProb**(InputArray **inputs**, OutputArray **outputs**, OutputArray **outputProbs**, int **flags**=0) → retval,

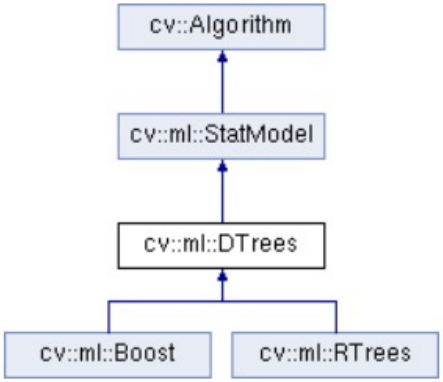
      results: output integer matrix of results. cv::Mat(m,1, **CV_32S**) (optional for single sample)

      outputProbs contains the output probabilities corresponding to each class of result. **cv::Mat(m,nc, CV_32F)**
      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 (float)

```
cv::Ptr<cv::ml::NormalBayesClassifier> bayes = cv::ml::NormalBayesClassifier::create();

bool retval = bayes->train(trainData, cv::ml::ROW_SAMPLE, trainLabels);
bayes->save("NormalBayes.yaml")            // Save trained classifier

// Test Normal Bayes (Prediction)
cv::Mat prediction, prob;
// bayes->predict(testData, prediction);
bayes->predictProb(testData, prediction, prob);
```

Reconocimiento de imágenes

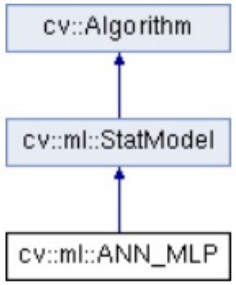


- Otros clasificadores: (módulo *machine learning*)
  - Clase: **cv::ml::DTrees** (Decision Tree)
    - cv::ml::Dtrees::**create**() → dtree (cv::**Ptr**<cv::ml::**Dtrees**>)
  - Métodos Dtrees:
    - cv::ml::Dtrees::**setCVFolds**(int **val**) val: (int) prune a tree with K-fold cross-validation
    - cv::ml::Dtrees::**setMaxDepth**(int **val**). val: (int) Max depth of the decision tree
    - cv::ml::Dtrees::**setTruncatePrunedTree**(bool **val**) val: (bool) # If true then a pruning will be harsher
    - cv::ml::Dtrees::**setUse1SERule**(bool **val**) val: (bool) If true pruned branches are removed from the tree
    - cv::ml::Dtrees::**setUseSurrogates**(bool **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 (float)

```
cv::Ptr<cv::ml::DTrees > 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
bool retval = dtree->train(trainData, cv::ml::ROW_SAMPLE, trainLabels);
dtree->save("DTree.yaml");             // Save trained classifier

// Test Decision Tre e(Prediction)
cv::Mat prediction;
dtree->predict(testData, prediction);
```

Reconocimiento de imágenes



- Otros clasificadores: (módulo *machine learning*)
  - Clase: **cv::ml::ANN_MLP** Neural Network – Perceptron Multicapa
    - cv::ml::ANN_MLP::**create**() → mlp (cv::**Ptr**<cv::ml::**ANN_MLP**>)

  - Métodos Dtrees:
    - cv::ml::ANN_MLP::**setActivationFunction**(**type** [, param1[, param2]])
      - **type**: cv::ml::ANN_MLP::IDENTITY | 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 | RPROP | ANNEAL
    - cv::ml::ANN_MLP::**setLayerSizes**(layer_sizes) DEBE CONFIGURARSE LA PRIMERA
      integer **cv::Mat** 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::**Ptr**<cv::ml::**ANN_MLP**>)

```
cv::Ptr<cv::ml::ANN_MLP > mlp = cv::ml::ANN_MLP::create();   // Pointer to the Specific Class  for parameters

// Set up MLP parameters
// FIRST layerSizes : integer vector with the number of neurons in each layer including the input and output layers.

cv::Mat layers = cv::Mat_<int>({ trainData.cols, 5, (int)labelRange.size()});
mlp->setLayerSizes(layers);  // 1 hidden layer  with 5 neurons

// Activation Function : cv::ml::ANN_MLP::IDENTITY | SIGMOID_SYM | GAUSSIAN | RELU | LEAKYRELU
// SIGMOID : param1 : alpha, param2 : beta
mlp->setActivationFunction(cv::ml::ANN_MLP::SIGMOID_SYM, 0.1, 1.5);

// Training methods : cv::ml::ANN_MLP::BACKPROP | RPROP | ANNEAL
mlp->setTrainMethod(cv::ml::ANN_MLP::BACKPROP);

mlp->setTermCriteria(cv::TermCriteria(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
  - Métodos:
    - cv::ml::StatModel::**train**(**samples**, **layout**, **responses**) → retval (bool)
    - cv::ml::StatModel::**predict**(**samples**, **results**, **flags**]) → retval (float)
    - **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
cv::Mat trainLabelsMat = cv::Mat::zeros(trainLabels.rows, labelRange.size(), CV_32F);
for (int row = 0; row < trainLabels.rows; row++)
        trainLabelsMat.at<float>(row, trainLabels.at<int>(row) - 1) = 1.0;

bool retval = mlp->train(trainData, cv::ml::ROW_SAMPLE, trainLabelsMat);
// Test MLP Neural Network(Prediction)
cv::Mat predictionMat;
mlp->predict(testData, predictionMat);

// convert predictionMat to mat vector : search for max response across classes(colums)
cv::Mat prediction;
for (int r = 0; r < predictionMat.rows; r++)
{
        cv::Point maxLoc; double maxVal;
        cv::minMaxLoc(predictionMat.row(r), 0, &maxVal, 0, &maxLoc);
        prediction.push_back((float)maxLoc.x + 1);
}
```

Reconocimiento de imágenes

- Otros clasificadores: (módulo *machine learning*)
  - Clase: **cv::ml::Boost** Adaboost
    - cv::ml::Boost::**create**() → classifier (cv::Ptr<cv::ml::Boost>)
  - Clase: **cv::ml::EM** Expectation/Maximization
    - cv::ml::EM::**create**() → classifier (cv::Ptr<cv::ml::EM>)
  - Clase: **cv::ml::KNearest** Vecino más cercano
    - cv::ml::Knearest::**create**() → classifier (cv::Ptr<cv::ml::Knearest>)
  - Clase: **cv::ml::LogisticRegression** Adaboost
    - cv::ml::LogisticRegression::**create**() → classifier (cv::Ptr<cv::ml::LogisticRegression>)
  - Clase: **cv::ml::RTrees** Multiple Random Trees
    - cv::ml::Rtrees::**create**() → classifier (cv::Ptr<cv::ml::cv.ml_Rtrees>)
  - Clase: **cv::ml::SVMSGD** Stochastic Gradient Descent SVM
    - cv::ml::SVMSGD::**create**() → classifier (cv::Ptr<cv::ml::SVMSGD>)

RECONOCIMIENTO 6b

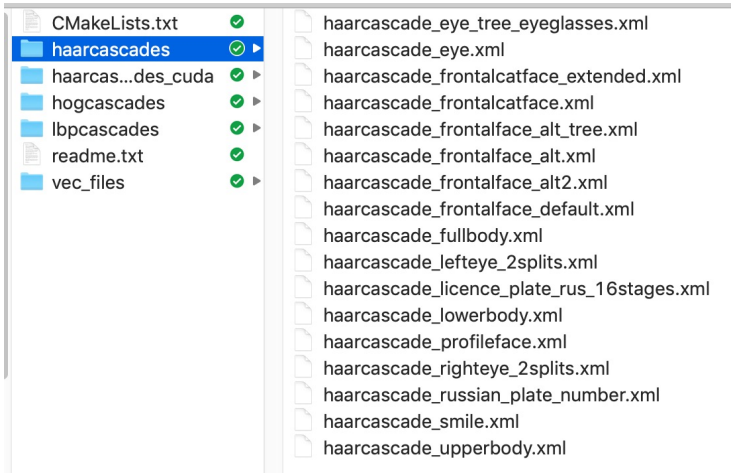
Módulo Object Detection (ej6b.cpp)

- 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: ej6b.cpp (módulo object detection)
  Detectar caras: (Haar//LBP → cv.CascadaClassifier)
  - Clase: **cv::CascadeClassifier**
    - cv::**CascadeClassifier**( ) → classifier <CascadeClassifier object>
    - cv::**CascadeClassifier**(string filename) → classifier <CascadeClassifier object>
      Carga datos del clasificador entrenado
  - Métodos:
    - cv::CascadeClassifier::**load**(string filename) → bool
      Loads a classifier from a file
    - cv::CascadeClassifier::**empty**() → bool
      Checks whether the classifier has been loaded



- cv::CascadeClassifier::**detectMultiScale**(InputArray **image**, vector< cv::Rect> & **objects**,
  double scaleFactor= 1.1, int minNeighbors=3, int flags=0,
  cv::Size minSize=cv::Size(), cv::Size maxSize=cv::Size())

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

Reconocimiento de imágenes

- Clasificadores en Cascada: ej6b.cpp (partimos del código de ej1.cpp)

```
string CASCADE_FACE_FILE = "../data/haarcascades/haarcascade_frontalface_alt.xml";
string CASCADE_EYES_FILE = "../data/haarcascades/haarcascade_eye_tree_eyeglasses.xml";

// Load cascade trained classifiers
cv::CascadeClassifier face_cascade, eyes_cascade;
face_cascade.load(CASCADE_FACE_FILE);
eyes_cascade.load(CASCADE_EYES_FILE);
```

```
cv::cvtColor(capture, gray_image, cv::COLOR_BGR2GRAY);    // transforms to gray level
cv::equalizeHist(gray_image, gray_image);  // Normalize gray levels
// Detect faces
vector<cv::Rect> faces;
face_cascade.detectMultiScale(gray_image, faces);
for (size_t i = 0; i < faces.size(); i++)
{
   cv::Point center(faces[i].x + faces[i].width / 2, faces[i].y + faces[i].height / 2);
   cv::ellipse(capture, center, cv::Size(faces[i].width / 2, faces[i].height / 2), 0, 0, 360, cv::Scalar(255, 0, 255), 4);

   cv::Mat faceROI = gray_image(faces[i]);
   vector<cv::Rect> eyes;  //-- In each face, detect eyes
   eyes_cascade.detectMultiScale(faceROI, eyes);
   for (size_t j = 0; j < eyes.size(); j++)
   {      cv::Point eye_center(faces[i].x + eyes[j].x + eyes[j].width / 2, faces[i].y + eyes[j].y + eyes[j].height / 2);
          int radius = cvRound((eyes[j].width + eyes[j].height)*0.25);
          cv::circle(capture, eye_center, radius, cv::Scalar(255, 0, 0), 4);
    }
}
```

Reconocimiento de imágenes

- Clasificadores en Cascada: ej6b.cpp (módulo object detection)
  Detectar peatones: (HOG → cv.HOGDescriptor) (SVM classifier)
  - Clase: **cv::HOGDescriptor**
    - cv::**HOGDescriptor**( ) → classifier <HOGDescriptor object>
    - cv::**HOGDescriptor**(string filename) → classifier <HOGDescriptor object>
      Carga datos del clasificador entrenado
  - Métodos:
    - cv::HOGDescriptor::**load**(string filename) → bool
      Loads a classifier from a file
    - cv::HOGDescriptor::**setSVMDetector** (svmdetector)
      svmdetector: cv::HOGDescriptor::**getDefaultPeopleDetector**()
      cv::HOGDescriptor::**getDaimlerPeopleDetector**()
    - cv::HOGDescriptor::**detectMultiScale**(InputArray **image**, vector< cv::Rect> & **objects,**
      vector<double> &**foundWeights,** double hitThreshold=0, c::Size winStride,
      c::Size padding, double scale=1.5, double groupThreshold=2.0, bool useMenshift=false)

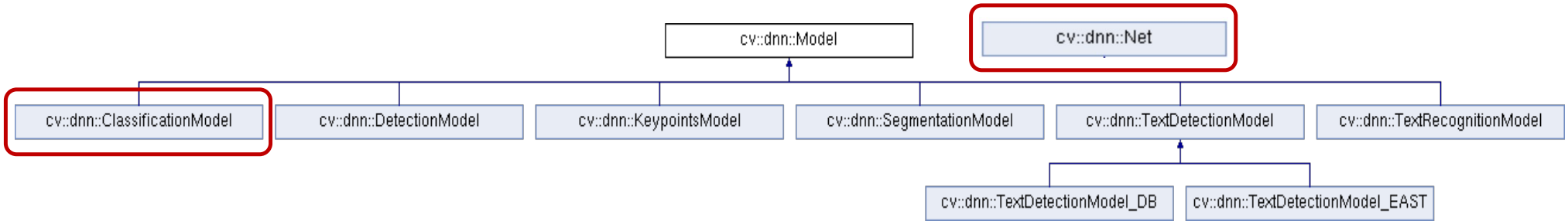
      foundLocations: vector cv::Rect(x, y, w, h) rectangle contains the detected object
      foundWeigths: vector<double> that will contain confidence values for each detected object

Reconocimiento de imágenes

- Clasificadores en Cascada: ej6b: HOG

```
// Configure pedestrian classifier
cv::HOGDescriptor pedestrian_detector;
pedestrian_detector.setSVMDetector(cv::HOGDescriptor::getDefaultPeopleDetector());
```

```
// Detect pedestrians
vector<cv::Rect> foundLocations;
vector<double> foundWeigths;
pedestrian_detector.detectMultiScale(gray_image, foundLocations, foundWeigths);

for (size_t i = 0; i < foundLocations.size(); i++)
{
        int x, y, w, h;
        x = foundLocations[i].x;  y = foundLocations[i].y;
        w = foundLocations[i].width; h = foundLocations[i].height;
        cv::rectangle(capture, cv::Point(x,y), cv::Point(x+w,y+h), cv::Scalar(0, 0, 200), 2);

        ostringstream text;
        text << setprecision(2) << foundWeigths[i];
        cv::Size textSize; int baseline;
        textSize = cv::getTextSize(text.str(), cv::FONT_HERSHEY_DUPLEX, 0.3, 1, &baseline);
                cv::rectangle(capture, cv::Point(x, y),  cv::Point(x + 10 + textSize.width, y - 10 - textSize.height),
                cv::Scalar(0, 0, 200), cv::FILLED);
        cv::putText(capture, text.str(), cv::Point(x + 5, y - 5), cv::FONT_HERSHEY_DUPLEX, 0.3,
                 cv::Scalar(0, 0, 0), 1, cv::LINE_AA);
}
```



REDES NEURONALES CONVOLUCIONALES (CNN)

Módulo Deep Learning (dnn) (ej6c.cpp)

- Importar redes pre-entrenadas (Caffe/Darknet)
- Clasificación: AlexNet, GoogLeNet
- Detección/Clasificación: YOLO

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

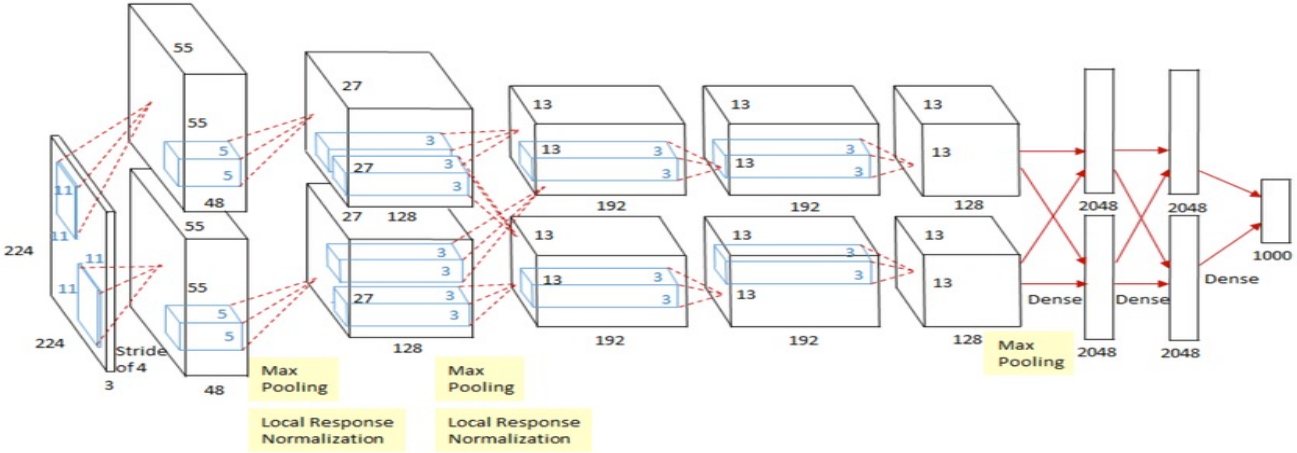
Deep Learning (CNN)

- 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)

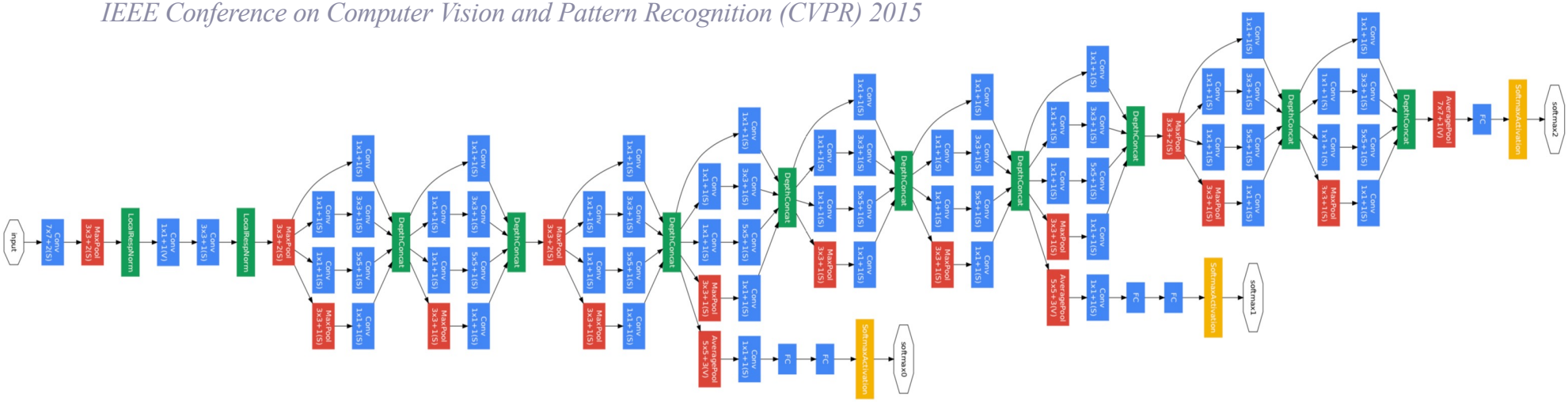
- **AlexNet:** (8/24 capas - 60Millones parámetros) - input 227x227x3

"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: **ej6c.cpp** (módulo Deep Learning)
  - Importar redes pre-entrenadas (Caffe)
- Clases: cv::dnn::Model, cv::dnn::ClassificationModel (OPCIÓN A)
  - cv::dnn::**Model**(cv::dnn:Net &**network**) → <Model object>
  - cv::dnn::**Model**(string **model**, string **config**) → <Model object>
  - cv::dnn::**ClassificationModel**(cv::dnn:Net **network**) → <ClassificationModel object>
  - cv::dnn::**ClassificationModel**(string **model**, string **config**) → <ClassificationModel object>

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

    config: (string) Nombre fichero de texto con la configuración de la red
    *.prototxt (**Caffe**), *.pbtxt (**TensorFlow**), *.cfg (**Darknet**), *.xml (**DLDT**)
- Métodos:
  - cv.dnn::Model::**setInputParams**(double **scale**, cv::Size **size**, cv::Scalar **mean**, bool **swapRB**, bool **crop**)
    blob(N, c, h, w) = scale * resize(frame(y, x, c)) - mean(c)) (Ver formato en crear blob cv::dnn::Net)
  - cv.dnn::Model::**predict**(InputArray **frame**, OutputArrayOfArrays **outs**)

    frame: Crea el blob de entrada, propaga (ejecuta) la red y devuelve los blobs de salida.
    outs vector<cv::Mat(1,nc)> blobs, almacena los resultados de los nodos de la capa de salida. (confianza - activación nodo)
  - cv::dnn::ClassificationModel::**classify**(InputArray **frame**) → std::**pair**<int, float> {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

    std::pair::**first**
    std::pair::**second**

Reconocimiento de imágenesEscriba aquí la ecuación.

- Programa: **ej6c.cpp** (módulo Deep Learning)
  - Importar redes pre-entrenadas (Caffe)
- Clase: cv::dnn::Net (OPCIÓN B)
  - cv::dnn::**readNet**(string **model**, string **config**, string **framework**) → <cv::dnn::Net object>

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

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

    framework: (string) Opcional, etiqueta con el nombre del Framework para determinar el formato
- Métodos:
  - cv::dnn::Net::**setInput**(InputArray **blob**, string **name**, double **scalefactor**, cv::Scalar **mean**)

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

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

    scalefactor: (double) Opcional, escala de normalización.

    mean: cv::Scalar Op. 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. cv::Mat(1,nc) (confianza)
  - cv::dnn::Net::**forward**(OutputArrayOfArrays &**outputBlobs**) vector<cv::Mat(1,nc)>
  - cv::dnn::Net::**empty**( ) → bool
  - cv::dnn::Net::**dump**( ) → retval (string). Crea cadena con la descripción de la red
  - cv::dnn::Net::**setPreferableBackend**(int backendId) Para elegir aceleración Hardware
  - cv::dnn::Net::**setPreferableTarget**(int targetId) También disponibles para la clase cv::dnn::Model

Reconocimiento de imágenes

- Programa: **ej6c.cpp** (módulo Deep Learning)
- Clase: cv.dnn_Net (OPCIÓN B)
- Funciones de formateo entradas de la red:
  - cv::dnn::**blobFromImage**(InputArray **image**, double **scale**, cv::Size **size**, cv::Scalar **mean**, bool **swapRB**, bool **crop** , int ddepth) → blob
  - cv::dnn::**blobFromImages**(**images** [, scalefactor[, size[, mean[, swapRB[, crop[, ddepth]]]]]]) → blob
  - cv::dnn::**imagesFromBlob**(cv::Mat **blob**, OutputArrayOfArrays **images**)

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

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

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

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

mean: (cv::Scalar) 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: (int) 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: **ej6c.cpp** (módulo Deep Learning)
- Clase: cv.dnn_Net (OPCIÓN B)
- Funciones de gestión de capas:
  - cv::dnn::Net::**getLayerNames**( ) → retval (vector<string>) lista de strings: nombre de capas
  - cv::dnn::Net::**getLayerId**(string layer) → retval (int) nombre capa a índice (int). (*empieza en 1*)
  - cv::dnn::Net::**getUnconnectedOutLayers**( ) → retval vector<int>: índices capas salida (*empieza en 1*)
  - cv::dnn::Net::**getUnconnectedOutLayersNames**( ) → retval (vector<string>) : nombre capas salida

  - cv::dnn::Net::**getLayerTypes**( ) → retval (vector<string>) : nombre tipos de capa
  - cv::dnn::Net::**getLayersCount**(layerType) → retval (int) numero capas por tipo (str)

- Búsqueda de máximo cv::Mat:
  - cv::**minMaxLoc** (InputArray **src**, double * **minVal**, double * **maxVal** = 0, cv::Point * **minLoc** = 0, cv::Point * **maxLoc** = 0, InputArray **mask** = noArray())
- 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: **ej6c.cpp** (módulo Deep Learning) (partiremos de **ej1.cpp**)
- Importar redes pre-entrenadas (Caffe)

```
string MODEL_FILE = "../caffe/bvlc_googlenet/bvlc_googlenet.caffemodel";
string CONFIG_FILE = "../caffe/bvlc_googlenet/deploy.prototxt";
string LABELS_FILE = "../caffe/bvlc_googlenet/imagenet-labels.txt";
cv::Size BLOB_SIZE(224, 224);          // input layer size

cv::TickMeter tm;            // TickMeter object to calculate FPS

// Load CNN data
cv::dnn::Net network = cv::dnn::readNet(MODEL_FILE, CONFIG_FILE, "Caffe");
cv::dnn::ClassificationModel networkModel(network);

// Load labels
vector<string> labels; string line;
ifstream fileHandler(LABELS_FILE.c_str());
while (getline(fileHandler, line))
if(!line.empty())
        labels.push_back(line);

// Show network model data
cout << network.dump() << endl;
cout << "#Classes: " << labels.size() << endl;
cout << "#Layers: " << network.getLayerNames().size() << endl;
cout << "#OutputLayers: " << network.getUnconnectedOutLayers().size() << endl;
```

Reconocimiento de imágenes

- Programa: **ej6c.cpp** (módulo Deep Learning)
- Usando la clase: cv.dnn_ClassificationModel (OPCIÓN A)

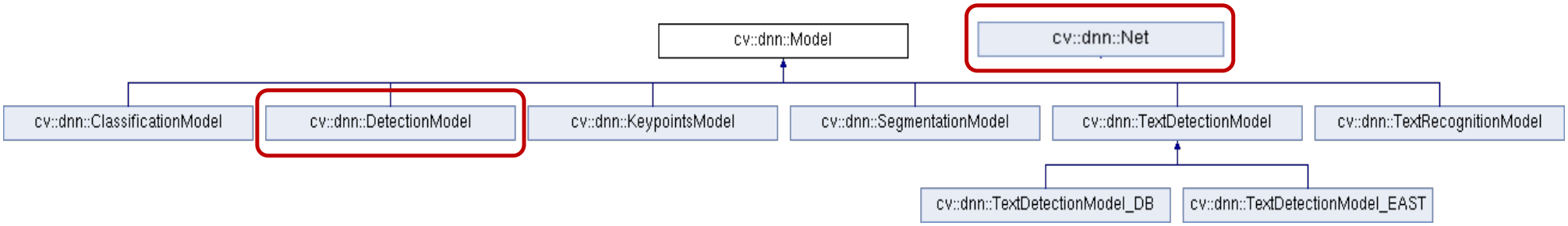
```
 ……..
 tm.start();  // start processing cycle

 //  image processing code using  cv::dnn::ClassificationModel
int classId; double conf;
networkModel.setInputParams(1.0, BLOB_SIZE, cv::mean(capture), true, true);

std::pair<int, float> res = networkModel.classify(capture);
classId = res.first;
conf = (double) res.second;

tm.stop();  // end processing cycle

// Show FPS on image
ostringstream text;
text << "FPS: " << setprecision(2) << tm.getFPS();
cv::putText(capture, text.str(), cv::Point(3, 20), cv::FONT_HERSHEY_DUPLEX, 0.5,
                    cv::Scalar(0, 0, 255),  1, cv::LINE_AA);
if (conf > 0.3)
{     ostringstream text;
      text << classId << " (" << setw(5) << setprecision(4) << conf * 100 << "%) - " << labels[classId];
      cv::putText(capture, text.str(), cv::Point(90, 20), cv::FONT_HERSHEY_DUPLEX, 0.5,
                    cv::Scalar(0, 265, 0), 1, cv::LINE_AA);
}
```

Reconocimiento de imágenes

- Programa: **ej6c.cpp** (módulo Deep Learning)
- Usando la clase cv.dnn_Net (OPCIÓN B)

```
 ……

 //  image processing code  using cv::dnn::Net
cv::Mat blob = cv::dnn::blobFromImage(capture, 1.0, BLOB_SIZE, cv::mean(capture), true, true);
network.setInput(blob);
cv::Mat outputBlobs = network.forward();

cv::Point classIdPoint;
cv::minMaxLoc( outputBlobs, 0, &conf, 0, &classIdPoint);
classId = classIdPoint.x;

……..
```

- 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) (ej6d.cpp)

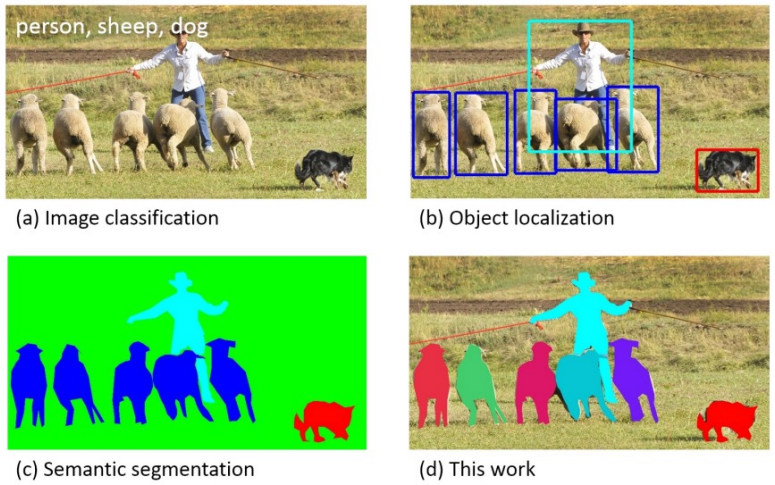
- Detección/Clasificación: YOLO

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) (416x416) ↑ 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 region
      - https://cocodataset.org



(a) Image classification (b) Object localization

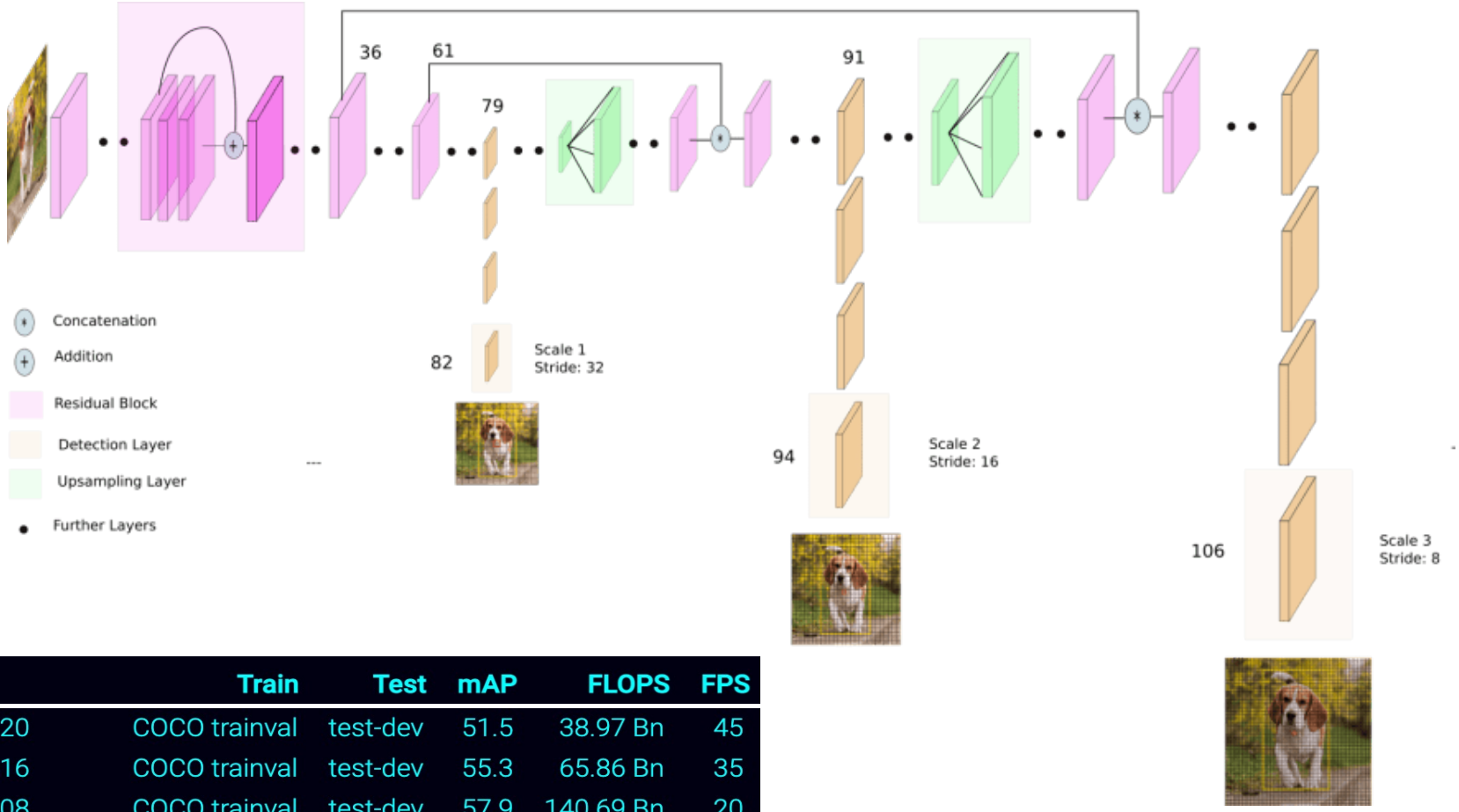
(c) Semantic segmentation (d) This work

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: **ej6d.cpp** (módulo Deep Learning)
  - Importar redes pre-entrenadas Darknet-YOLO

- Clases: cv.dnn_DetectionModel (YOLO9000 no funciona con las clases de OpenCV)
  - cv::dnn::**DetectionModel**(cv::dnn:Net &**network**) → <DetectionModel object>
  - cv::dnn::**DetectionModel**(string **model**, string **config**) → <DetectionModel object>

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

  - cv::dnn::DetectionModel::**detect**(InputArray **frame**, vector<int> **classIds**, vector<float> **confidences**, vector<cv::Rect> **boxes**, float **confThreshold**=0.5, float **nmsThreshold**=0.0)

    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: vector<int>
    confidences: vector<float> valor de probabilidad
    boxes: vector<cv::Rect> (x,y,w,h). Conjunto de ventanas

  - cv::dnn::DetectionModel::**setNmsAcrossClasses**(bool value)

Reconocimiento de imágenes

- Programa: **ej6d.cpp** (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. cv::Mat(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**(OutputArrayOfArrays &**outputBlobs**, vector<string> **outputNames**)
    outputBlobs : vector<cv:Mat(1,nc)> (N=1 para 1 imagen)
    outNames: 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 (**bboxes** vector<cv::Rect >) solapadas, con su correspondiente confianza (**scores** vector<float>). Se queda con la de mayor probabilidad

Reconocimiento de imágenes

- Programa: **ej6d.cpp** (módulo Deep Learning) (partiremos de **ej6c.cpp**)
- Importar redes pre-entrenadas (Darknet-YOLO)

```
string MODEL_FILE = "../darknet/yolov3.weights";
string CONFIG_FILE = "../darknet/yolov3.cfg";
string LABELS_FILE = "../darknet/coco.names";
cv::Size BLOB_SIZE(320, 320);          // downsized input layer size (increase FPS)
tm = cv.TickMeter();                   // TickMeter object to calculate FPS

# Load CNN data
cv::dnn::Net network = cv::dnn::readNet(MODEL_FILE, CONFIG_FILE, "Darknet")
cv::dnn::DetectionModel  networkModel(network);

// Load labels
vector<string> labels; string line;
ifstream fileHandler(LABELS_FILE.c_str());
while (getline(fileHandler, line))
if(!line.empty())
        labels.push_back(line);

// create a list of random colors for each label
vector<cv::Scalar> colors; cv::RNG rng;       // random number generator
for (int i = 0; i< labels.size(); i++)
        colors.push_back(cv::Scalar(rng.uniform(0, 256), rng.uniform(0, 256), rng.uniform(0, 256)));

# Increase camera resolution
camera.set(cv::CAP_PROP_FRAME_WIDTH, 960)
camera.set(cv::CAP_PROP_FRAME_HEIGHT, 720)
```

Reconocimiento de imágenes

- Programa: **ej6d.cpp** (módulo Deep Learning)
- Procesamiento clase cv::dnn::ClassificationModel (Darknet-YOLO)

```
tm.start();  // start processing cycle
// Important: input blob color levels must be scaled to [0-1]
networkModel.setInputParams(1.0/255, BLOB_SIZE, cv::mean(capture), true, false);
networkModel.setNmsAcrossClasses(false);  // False : NMS only for bboxes of the same class
networkModel.detect(capture, classIds, confidences, boxes, 0.3, 0.2);

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

```
// Show Detected Bounding Boxes
for  (int i = 0; i < classIds.size(); i++)
{
      int classId = classIds[i];
      float confidence = confidences[i];
      int x = boxes[i].x, y = boxes[i].y, w = boxes[i].width, h = boxes[i].height;
      cv::rectangle(capture, cv::Point(x, y), cv::Point(x + w, y + h), colors[classId], 2);

      ostringstream text;
      text << labels[classId] << ": " << setprecision(4) << confidence * 100  << "%";
      cv::Size textSize; int baseline;
      textSize = cv::getTextSize(text.str(), cv::FONT_HERSHEY_DUPLEX, 0.3, 1, &baseline);
      cv::rectangle(capture, cv::Point(x, y), cv::Point(x + 10 + textSize.width, y - 10 - textSize.height),
                              colors[classId], cv::FILLED);
      cv::putText(capture, text.str(), cv::Point(x + 5, y - 5), cv::FONT_HERSHEY_DUPLEX, 0.3,
                              cv::Scalar(0, 0, 0), 1, cv::LINE_AA);
}
```