Homework #1: Model Optimization for Face Detection

Every student will need to submit their own work. Do not share code. All submissions will be cross-checked against each other.

1 Face Detection

The goal of this assignment is to develop a face detection method that works for multiple scales and different aspect ratios. The basic approach is based on generating proposal boxes, processing them through PCA and SVM and optimizing the pipelined model.

The primary reference for the assignment is: *Introduction to Machine Learning with Python* by Sarah Guido, Andreas C. Müller, O'Reilly Media, Inc., Oct. 2016. The material is drawn from the following chapters:

```
2. Supervised Learning
3. Unsupervised Learning and Preprocessing
4. Representing Data and Engineering Features
5. Model Evaluation and Improvement
6. Algorithm Chains and Pipelines
```

From UNM, you can access the book using `https://libguides.unm.edu/Safari`. If you need to read the book offline, you can also download individual chapters. However, if you do that, note that the web links do not work.

The assignment is broken into the following steps:

1. **Generate ground truth on images:** The approach here is to download the AOLME videos and select at-least ten images of three faces from the videos. You will need to place a rectangle along each face region.

   You can download the videos from `http://ivpcl.unm.edu/ivpclpages/Research/videoactivityrecognition/index.php`:

   ```
   Scroll down to "Non-IRB Material"
   Look for the button identified as "Download videos".
   ```

   You will need a high-speed interface to download 442 MB. Please review the file `ROIdemo.py` that contains the code for generating ground truth and transforming each image to grayscale. You are free to use the Matlab video labeler if you prefer using Matlab.

   You will need to generate 3 face sub-videos for 10 different videos.

2. **Use image ground truth to generate ground truth for boxes** Use the ground truth on the face images to generate ground truth for each box size.

   You will need to train four classifiers. To store the different options, you are asked to use the dictionary `BoxesInfoDict` with the following members:

   ```
     "Scales"       = [1, 1, 2, 2]
     "AspectRatios" = [1, 2, 1, 2]
   ```

   We will basically train one classifier for each combination of `Scale` and `AspectRatio`.

   The boxes are generated by spacing `AnchorPoints`, `BoxRows`, and `BoxCols`:

   ```
     Set AnchorPoints using a regular 2D grid to generate
         (X, Y) coordinates that are Spacing pixels
         apart among rows and columns.
   ```

```
 BoxCols = Scale*UnitScaleSpacing
 BoxRows = AspectRatio*NumOfCols
```

Please note that the same boxes are also generated in the function called `ClassifyImage()`.

For each image and box size, you can call `BuildTrainData()` to create ground truth. The following code shows the steps for building ground truth per frame.

```
FaceBoxes  = []
BackBoxes  = []
FrameGroup = []
FrameCtr   = 0
For each video:
         Select one frame every 10 frames in the video
         Set FaceCoords for the image
         NumOfBoxes = 6 # 3 large face boxes and backgrounds
         [NewFaceBoxes, NewBackBoxes] = BuildTrainData(
               FaceCoords, AnchorPoints, BoxRows, BoxCols,
               Th, NumOfBoxes)
         FaceBoxes.append(NewFaceBoxes)
         BackBoxes.append(NewBackBoxes)
         FrameGroup.append(FrameCtr)
         FrameCtr = FrameCtr + 1

X = FaceBoxes + BackBoxes  with one box rerpresented per row
y = list of 1s for the faces followed by 0s for the backgrounds
```

```
def BuildTrainData (img, FaceCoords, AnchorPoints,
                  BoxRows, BoxCols, Th, NumOfBoxes)
       """
       Generates an equal number of Faces and non-Face boxes
       from the given image data.

       img          contains a single image frame.
       FaceCoords   contains the coordinates for all faces.
       AnchorPoints contains the central coordinates for all boxes.
       BoxRows      is the number of rows for each box.
       BoxCols      is the number if columns for each box.
       Th           is a threshold to determine if we
                    have a face box or not.
       NumOfBoxes   is an even number of the number
                    of boxes to produce.
       """
       NumOfFaceBoxes    = 0
       NumofNonFaceBoxes = 0
       for sample=1 to MaxSamples:
          Randomly sample a box from the list of AnchorPoints
          Build a box of size BoxRows and BoxCols centered
                at the sampled anchor point

          Apply BoxGroundTruth() to establish if the box
                significantly overlaps a face or not

          if (box is a face  and
                  (NumOfFaceBoxes < NumOfBoxes/2))
                Append box of img pixels to face boxes
                NumOfFaceBoxes = NumOfFaceBoxes+1
```

```

            else if (NumOfNonFaceBoxes < NumOfBoxes/2)
                  Append box of img pixels to non-face boxes
                  NumOfNonFaceBoxes = NumOfNonFaceBoxes+1

            If (NumOfFaceBoxes+NumofNonFace == NumOfBoxes)
                   Break out of the loop

            # IGNORE ORIGINAL SUGGESTION FROM THE PAPER
            # RemainingBoxes = NumOfBoxes-NumOfFaceBoxes-NumOfNonFaceBoxes
            # if (RemainingBoxes>0)
            #    Sample the RemainingBoxes from the Non-face-boxes

            d = NumOfNonFaceBoxes - NumOfFaceBoxes
            If (d>0)
                Remove d non-face boxes from their list
            Else
                Remove -d face boxes from their list


        Return the equal lists of face-boxes and non-face-boxes
```

Here, I also provide pseudocode for determining whether a box sufficiently overlaps a face to be classied as a face box:

```
def BoxGroundTruth(BoxCoords, FaceCoords, thresh)
       """
       Returns True if BoxCoords overlaps a face box and
       OverlapRegion / UnionRegion > thresh

       BoxCoords  = contains the coordinates of a single box.
       FaceCoords = contains the coordinates of all of
       the faces within the current image.
       """
       ...
```

For the intersection-over-union ratio, we note that it is bounded by:

$$\text{IoU} = \frac{\text{Area(Box } \textbf{and} \text{ Face pixels)}}{\text{Area(Box } \textbf{or} \text{ Face pixels)}} \leq \frac{\min(\text{Area(Face pixels)}, \text{Area(Box)})}{\text{Area(Box } \textbf{or} \text{ Face pixels)}} = T_{\max}.$$

Thus, you need to set your threshold between 0 and $T_{\max}$. We also mentioned in class that it is best to have the boxes be large enough to capture the face inside.

You will need to generate ground truth of at-least 50 images per face (3 faces with 50 images each). Furthermore, as we discussed in class, you will need to only use one every N images (e.g., one every 30 images).

3. **Optimized model performance estimation** The goal of this last step is to measure the performance of the optimized model. The basic processing model is given in `plot_face_recognition.ipynb`. After you study the code in `plot_face_recognition.ipynb`, you will need to modify the `BoxClassifier()` code given below. The code below provides source code for nested cross-validation (similar to `ML-CV.ipynb`). The optimization approach is compatible with our pipeline example given in `ML-Pipeline-Ex.ipynb`).

```
# Class for implementing your classifier for SciKit Learn.
# Initial code taken from:
#   http://danielhnyk.cz/creating-your-own-estimator-scikit-learn/
from sklearn.base import BaseEstimator, ClassifierMixin

class BoxClassifier(BaseEstimator, ClassifierMixin):
    """An example of classifier"""

    # Initialize by passing the parameters of your model:
    def __init__(self, NumOfPCAcomponents, C, gamma):
        """
        Called when initializing the classifier.
        """
        self.C_ = C
        self.gamma_ = gamma
        self.NumOfPCAcomponents_ = NumOfPCAcomponents

    # Please add PCA and SVM in this template.
    # Make sure to use the given parameters.
    def fit(self, X, y):
        """
        This should fit classifier. All the "work" should be done here.
        """
        ... call .fit() for PCA and .fit for SVC()
        return self

    def predict(self, X):
        """
        Predicts the classification of a single box of pixels.
        """
        ... apply PCA and .predict() for SVC()
        ... Deterimine Class_result for SVC.
        return(Class_result)

    def score(self, X, y):
        """
         Returns the classification accuracy assuming
         balanced datasets (half in each category).
        """
        ... Applies predition on X, and compares the results
        ... against the actual values in y.
        return(accuracy)

def nested_cv(X, y, groups, inner_cv, outer_cv, Classifier, parameter_grid):
    """
    Uses nested cross-validation to optimize and exhaustively evaluate
    the performance of a given classifier. The original code was taken from
    Chapter 5 of Introduction to Machine Learning with Python. However, it
    has been modified.

    Input parameters:
       X, y, groups: describe one set of boxes grouped by image number.

    Output:
       The function returns the scores from the outer loop.
    """
    outer_scores = []
    # for each split of the data in the outer cross-validation
    # (split method returns indices of training and test parts)
    #
    for training_samples, test_samples in outer_cv.split(X, y, groups):
        # find best parameter using inner cross-validation
        best_parms = {}
        best_score = -np.inf
        # iterate over parameters
        for parameters in parameter_grid:
            # accumulate score over inner splits
            cv_scores = []
            # iterate over inner cross-validation
            for inner_train, inner_test in inner_cv.split(
                      X[training_samples], y[training_samples],
                      groups[training_samples]):
                      # build classifier given parameters and training data
                      clf = Classifier(**parameters)
                      clf.fit(X[inner_train], y[inner_train])

                      # evaluate on inner test set
                      score = clf.score(X[inner_test], y[inner_test])
                      cv_scores.append(score)

            # compute mean score over inner folds
            # for a single combination of parameters.
            mean_score = np.mean(cv_scores)
            if mean_score > best_score:
                # if better than so far, remember parameters
                best_score = mean_score
                best_params = parameters

        # Build classifier on best parameters using outer training set
        # This is done over all parameters evaluated through a single
        # outer fold and all inner folds.
        clf = Classifier(**best_params)
        clf.fit(X[training_samples], y[training_samples])

        # evaluate
        outer_scores.append(clf.score(X[test_samples], y[test_samples]))
    return np.array(outer_scores)


from sklearn.model_selection import ParameterGrid, GroupKFold
# You need to carefully initialize X and y as described previously.
# Furthermore, inner_cv and outer_cv must be initialized using GroupKFold.
# In the parameter_grid, you will need to pass SVM parameters and the
# number of PCA components.
parameter_grid = ParameterGrid(DictionaryOfValues)
scores = nested_cv(X, y, groups, inner_cv, outer_cv, Classifier, parameter_grid)
print("Cross-validation scores: {}".format(scores))
```

In order to setup the parameter grid using a dictionary, refer to `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.ParameterGrid.html`. You can use a dictionary and then use `ParameterGrid()` to combine all of the tuples together for the parameter for-loop:

```
parameter_grid = ParameterGrid(DictionaryOfValues)
scores = nested_cv(X, y, groups, inner_cv, outer_cv,
                   Classifier, parameter_grid)
```

Note that you will have to generate four classifiers. One for each box size.

Note that `inner_cv` and `outer_cv` will need to be set using `KFold()` or `StratifiedKFold()` (more general). More information is available at `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html`. As an example, the following code provided by a student, supports `predictable random shuffling`:

```python
inner_cv = KFold(n_splits=5, shuffle=True, random_state=0)
outer_cv = KFold(n_splits=5, shuffle=True, random_state=0)
```

4. **Optimize the final model and obtain results from the final classifier** The final classifier will have to be applicable to images directly. After training and cross-validation, there is still a need to develop the final classifier that operates on entire images, not just boxes. This section provides the pseudocode for the final pseudocode that will be run on images.

   For each one of the four classifier combinations, you need to call the `ClassifyImage()` method given below **(also attached in the file `nestedCV.py`)**:

```
def ClassifyImage(InputDict, FaceBoxes):
 """
 Generates boxes for a single image and a single classifier.

  InputDict members:
    "img"            = Input image.
    "Spacing"        = Spacing among anchor points
    "UnitScaleSize"  = Box size for scale=1.
    "Scale"          = Scale for box detection.
    "AspectRatio"    = Contains the aspect ratio for the box.
    "Classifier"     = Contains the PCA+SVM classification pipeline.

  Output:
    FaceBoxes = Contains the coordinates of the boxes
                classified as faces.
  """

  DX = Spacing   # Set up the Anchor point spacing.
  Dy = Dx
  for X = Dx to LenX-Dx
      for Y = Dy to LexY-Dy
         Use BoxesInfoDict to generate
         a single proposal box using:
           NumOfCols = Scale*UnitScaleSpacing
           NumOfRows = AspectRatio*NumOfCols
           centered at (X, Y)
           and extracted from img

         Classify each box using the given classifier.

         If the classified box is a face:
             append it to FaceBoxes
```

   Train the final classifier using 80% of the total training data and 20% of the rest of the data for validation. Add example runs on three different images.

2 Deliverables

Provide brief summaries with results from the following:

- Python code for all of the steps.
- Ground truth process including screenshots showing ground boxes.
- Box generation including screenshots of boxes placed on images.
- Optimized model performance by displaying the accuracy values achieved.
- Final results displayed per frame.

3 Python Editor: Installation of Spyder

During the demo in class, the student used the `Spyder` Python editor. If you want to install and run `Spyder`, open up the Anaconda Prompt and type:

```
activate tf2
conda install spyder
spyder
```

4 Additional References from SciKit Learn Website

The following links have tutorials on model optimization using SciKit Learn:

- General references for Cross-validation.
- Estimate confidence intervals for ROC.
- Nice way to visualize how cross-validation is done.
- General Grid Search reference.
- Best way to evaluate performance is based on nested cross-validation.
- Optimize BOTH AUC and accuracy.
- Randomized cross-validation versus exhaustive evaluation.
- Balance model complexity and cross-validation score.

5 RCNN Papers

Attached to your assignment, you will find the three papers that motivated your assignment. By the end of the class, you should be able to understand all aspects of these papers.