# Computer Vision Project: Dexter's Laboratory

Lungu Laura-Vanesa

January 21, 2025

## Introduction

This project focuses on implementing a system for automated facial detection and recognition of characters from the animated series *Dexter's Laboratory*. The task is divided into two main objectives:

- **Facial Detection**: Accurately identify and localize all character faces in images using bounding boxes and confidence scores.
- **Facial Recognition**: Classify detected faces into one of four main characters—Dexter, DeeDee, Mom, or Dad.

## 1 Solution Overview

The solution employs classical computer vision techniques to address the task of facial detection. It uses:

- **Feature extraction**: Histograms of Oriented Gradients (HOG) are computed to capture meaningful facial patterns.
- **Classification**: Linear Support Vector Machines (SVM) are trained to differentiate between faces and non-face regions.

The system is evaluated using:

- **Precision-recall graphs**: To visualize the trade-off between precision and recall.
- **Metrics**: Average Precision (AP) and mean Average Precision (mAP) provide quantitative assessments of performance against ground truth annotations.

These techniques ensure a robust and interpretable framework for facial detection.

## 2 Image Preprocessing

The function `get_positive_and_negative_examples()` is designed to prepare the training data by processing:

- **Positive examples**: Crops containing faces from annotated images.
- **Negative examples**: Randomly sampled non-face regions.

### 2.1 Positive Example Processing

The function iterates through training images and their bounding box annotations. For each face:

1. **Cropping**: Extracts the face region with a 10-pixel margin around the bounding box to include some context.

2. **Aspect Ratio Analysis**: Determines the shape of the cropped face:
    - **Horizontal Rectangle**: Width is significantly greater than height (e.g., DeeDee's face; $w/h > 1.2$).
    - **Vertical Rectangle**: Height is significantly greater than width (e.g., Dad's face; $w/h < 0.8$).
    - **Square**: Width and height are roughly equal ($0.8 \leq w/h \leq 1.2$).
3. **Saving Crops**: Resizes the cropped faces to predefined dimensions and organizes them into directories based on:
    - **Character**: (e.g., `dad`, `deedee`).
    - **Shape type**: (e.g., `horizontal`, `vertical`, `square`).
4. **Augmentation**: Enhances the dataset using transformations from the `albumentations` library:
    - `HorizontalFlip`: Flips the image horizontally (`p=1.0`).
    - `ShiftScaleRotate`: Applies random shifts, scaling, and rotations (`shift_limit=0.05`, `scale_limit=0.1`, `rotate_limit=45`, `p=1.0`).
    - `Perspective`: Applies a random perspective transformation with scaling between 0.05 and 0.1 (`keep_size=True`, `pad_mode=0`, `p=1.0`).

### 2.2 Negative Example Processing

The function also generates negative examples by randomly cropping 96x96 regions from training images that do not overlap any face bounding boxes.

This preprocessing step ensures a structured dataset for training the HOG-based classifier.

## 3 HOG Descriptors

Histograms of Oriented Gradients (HOG) are feature descriptors commonly used in computer vision tasks to capture the structural and gradient information of an image. They are particularly effective for detecting objects such as faces due to their ability to encode edges and shapes.

### 3.1 What Are HOG Descriptors?

HOG descriptors work by dividing an image into small connected regions called **cells**. For each cell:

1. The gradient (change in intensity) is calculated for every pixel.
2. These gradients are binned into predefined **orientations** (e.g., 9 bins representing angles between 0° and 180°).
3. The histogram of these orientations is normalized over larger regions called **blocks**, which ensures the descriptor is invariant to changes in lighting or contrast.

The final HOG descriptor is a concatenation of all block histograms, resulting in a feature vector that represents the image.

### 3.2 HOG Descriptor Creation for This Project

In this project, HOG descriptors are generated for both positive (face) and negative (non-face) examples. The process is dimension-specific to accommodate the unique characteristics of the faces in the dataset, categorized into: horizontal, vertical, and square. Before computing the HOG descriptors, all images undergo a preprocessing pipeline to enhance their quality and consistency.

This preprocessing includes:

- **Histogram Equalization (CLAHE)**: Improves contrast by redistributing pixel intensity.
- **Gamma Correction**: Adjusts brightness levels for consistent illumination.
- **Gaussian Blur**: Reduces noise while preserving important features like edges.
- **Normalization**: Scales pixel values to the range [0, 1] for uniform processing.
- **Resizing**: Adjusts the image to the target dimensions required for HOG calculation.

These transformations are implemented using the `albumentations` library to ensure robust and efficient image processing.

The HOG descriptor generation process for each dimension proceeds as follows:

1. The cropped images are resized to match the target window dimensions:
   - **Horizontal**: $96 \times 84$ pixels.
   - **Vertical**: $84 \times 96$ pixels.
   - **Square**: $96 \times 96$ pixels.
2. HOG parameters are configured as follows:
   - **Cell size**: $6 \times 6$ pixels.
   - **Block size**: $2 \times 2$ cells.
   - **Orientations**: 9 gradient directions.
3. Each image is preprocessed using the `preprocess_image()` function, which applies the aforementioned transformations.
4. The HOG descriptors are computed for the preprocessed images using the `get_positive_descriptors()` and `get_negative_descriptors()` functions.

By applying preprocessing before descriptor calculation, the system ensures that the extracted HOG features are consistent, robust, and less sensitive to variations in lighting or noise. This step is critical for achieving high accuracy in face detection.

# 4 Model Training

Once the HOG descriptors are generated, the system trains three separate models—one for each face dimension: **horizontal**, **vertical**, and **square**. Each model is optimized to classify between faces and non-faces based on the specific characteristics of that dimension.

## 4.1 Training Process

The training process is as follows:

1. **Data Preparation**:
   - Positive and negative HOG descriptors are loaded for each dimension from the precomputed files.
   - Positive examples are labeled as 1, and negative examples are labeled as 0.
   - The combined data forms the training dataset for the classifier.
2. **Classifier Training**:
   - A **Linear Support Vector Machine (SVM)** is trained for each dimension using the `train_classifier()` function.

- The regularization parameter $C$ is tuned over a range of values ($10^{-5}$ to $10^{0}$) to find the optimal balance between margin maximization and misclassification.
- The model with the highest accuracy on the training data is selected.

3. **Model Saving**:
   - Each trained model is saved to disk for later use during the detection phase (e.g., `task1_model_horizontal.pkl`).

# 5 Detection and Evaluation

After training the models, the final step involves running the detection system on test images and evaluating its performance. This phase combines sliding window detection, multi-scale analysis, and evaluation metrics to ensure accurate and interpretable results.

## 5.1 Detection Process

The detection phase leverages the trained models to identify faces in the test images. The process is as follows:

1. **Image Preprocessing**:
   - Test images are loaded from the directory specified in `params.dir_test_examples`.
   - Images are converted to grayscale for consistent feature extraction and efficiency.
2. **Multi-Scale Detection**:
   - Each image is resized across multiple scales (starting from 0.8 and progressively reduced by a factor of 0.8).
   - This ensures that faces of varying sizes can be detected within the same image.
3. **Sliding Window Analysis**:
   - A sliding window scans the image at each scale, extracting HOG descriptors for the window.
   - These descriptors are fed into the corresponding trained model (horizontal, vertical, or square).
   - Windows with confidence scores above a defined threshold are considered potential detections.
4. **Non-Maximal Suppression (NMS)**:
   - To handle overlapping detections, NMS is applied to retain only the most confident bounding box for each face.

## 5.2 Evaluation

The final step involves evaluating the system's performance on the test dataset. The trained models were used to detect faces in the test images, and the results were assessed using standard evaluation metrics.

### 5.2.1 Metrics and Results

The evaluation focuses on the following metrics:

- **Precision**: The proportion of correct detections among all detections.
- **Recall**: The proportion of actual faces that were detected.
- **Average Precision (AP)**: Summarizes precision-recall performance across different confidence thresholds.

On the test dataset, the system achieved an **Average Precision (AP)** of **57%**, which demonstrates the model's capability to detect faces with reasonable accuracy.

### 5.2.2 Precision-Recall Curve

Figure 1 shows the precision-recall curve generated during the evaluation phase. The curve highlights the trade-off between precision and recall across different confidence thresholds.

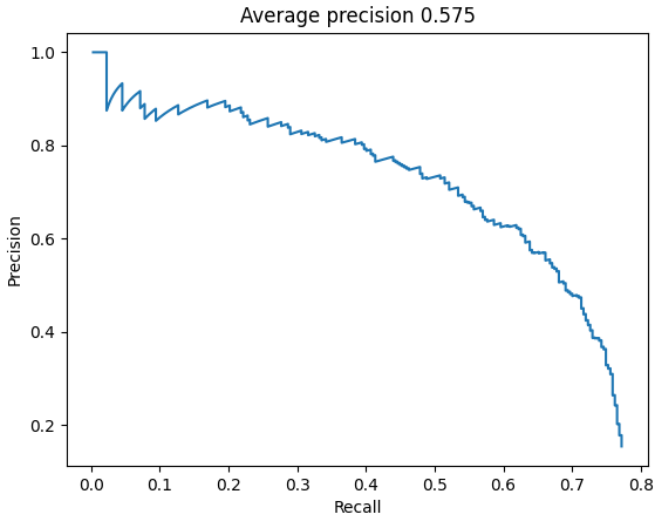


Figure 1: Precision-Recall curve on the test dataset. The model achieved an AP of 57%.

### 5.2.3 Example Detections

Figure 2 illustrates some example detections produced by the system. Detected faces are marked with bounding boxes, and their confidence scores are displayed. This visualization helps validate the model's ability to localize faces effectively.

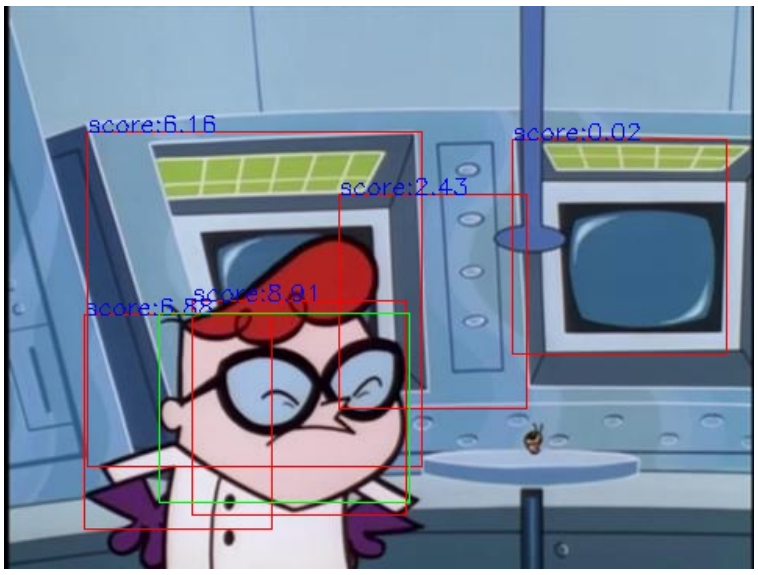


Figure 2: Example detections from the test dataset.