Computer Vision Project

Object Detection System Documentation

1. Introduction

1.1. Project Objective

The primary objective of this project, is to design and implement a complete object detection system focused on a specific field. The system is required to include a (GUI) capable of accepting a static image, detecting relevant objects within that image.

1.2. System Overview

The implemented system provides a user-friendly graphical interface built with Gradio. A user can upload an image containing an animal. The system then processes the image through a pipeline involving preprocessing, segmentation (to isolate the main subject), feature extraction (using Histogram of Oriented Gradients - HOG), and classification (using a Multi-Layer Perceptron - MLP trained on HOG features).

---

2. Dataset

2.1. Dataset Name & Source

- Name: Animals-10
- Source: Kaggle
- URL: https://www.kaggle.com/datasets/alessiocorrado99/animals10
- Citation (from Kaggle): Corrado, Alessio. (2016). Animals-10 [Data set]. Kaggle. https://doi.org/10.34740/KAGGLE/DSV/846261

2.2. Dataset Description

The Animals-10 dataset contains images belonging to 10 different categories of animals in Italian.

- Number of Classes: 10
- **Classes (Original Italian names translated to English in the code) :**
    1. dog (cane)
    2. horse (cavallo)

3. elephant (elefante)
4. butterfly (farfalla)
5. chicken (gallina)
6. cat (gatto)
7. cow (mucca)
8. sheep (pecora)
9. spider (ragno)
10. squirrel (scoiattolo)

Total Images (after augmentation & split): The code splits the dataset into training (80%), validation (10%), and testing (10%) sets after augmentation. This results in approximately:

Training: ~38,900 images (3890 per class)

Validation: ~4,860 images (486 per class)

Testing: ~4,870 images (487 per class)

2.3. Rationale for Selection

This dataset was selected for the implementation as it provides a readily available, and it's number of classes are matching our situation, multi-class image classification task suitable for demonstrating the computer vision pipeline, including preprocessing, feature extraction, and classification techniques.

---

3. System Architecture & Pipeline

3.1. Pipeline Step 1: Preprocessing & Image Enhancement

This step aims to prepare the input images for subsequent processing, making them more suitable for segmentation and feature extraction.

- **Techniques Used:**
    - **Offline (Data Preparation):**

  - ***Data Augmentation (ImageDataGenerator):*** Applied to balance the dataset for classes with fewer images. Included rotation (±20 degrees), zoom (±20%), brightness adjustment (70-130%), and horizontal flips.
  - ***Resizing:*** Images were resized to 224x224 during augmentation generation.
- **Online (Applied to Train/Val/Test sets before Segmentation/Feature Extraction via custom_preprocessing function):**
  - ***Color Conversion (BGR to RGB):*** Ensures consistent color channel order (using cv2.cvtColor).
  - ***Normalization:*** Pixel values are scaled to the [0, 1] range by dividing by 255.0.
  - ***Gaussian Blur:*** A 5x5 kernel is applied (cv2.GaussianBlur) to reduce noise.
  - ***Contrast Limited Adaptive Histogram Equalization (CLAHE):*** Applied to the L-channel in the LAB color space (cv2.createCLAHE, cv2.cvtColor) to enhance local contrast without over-amplifying noise.
  - ***Sharpening:*** A 3x3 sharpening kernel (cv2.filter2D) is applied to enhance edges.

During Training/Inference Data Loading (`ImageDataGenerator` for training, separate one for val/test):

Rescaling: Pixel values are scaled by 1/255.

Training Augmentations: Further augmentations (rotation, shifts, zoom, brightness, shear, flip) are applied on-the-fly to the training data via `ImageDataGenerator`.

What is the steps?

1) Normalization ensures pixel values are within a consistent range, aiding model training stability.
2) Blurring helps reduce image noise that might interfere with feature extraction.
3) CLAHE improves image contrast locally, making features more distinct, especially in varying lighting conditions.
4) Sharpening enhances edges, which can be important for HOG features.

5) Data augmentation (offline and online) increases the diversity of the training data, helping the model generalize better and reduce overfitting.

3.3. Pipeline Step 2: Segmentation / Region Proposal:

- Techniques Used (thresholding_segmentation function):

    1. ***Grayscale Conversion****:* The preprocessed image is converted to grayscale (cv2.cvtColor).
    2. ***Binary Thresholding****:* A fixed threshold (value 127) is applied (cv2.threshold with THRESH_BINARY) to create a binary mask.
    3. ***Morphological Closing****:* A 5x5 kernel is used (cv2.morphologyEx with MORPH_CLOSE) to close small holes and gaps in the binary mask, aiming to create a more solid representation of the object.
    4. ***Contour Detection****:* External contours are found in the closed mask (cv2.findContours with RETR_EXTERNAL).
    5. ***Largest Contour Selection****:* The contour with the largest area is selected, assuming it corresponds to the main subject.
    6. ***Bounding Box Extraction****:* The bounding box of the largest contour is calculated (cv2.boundingRect).
    7. ***ROI Cropping & Resizing****:* The region defined by the bounding box is cropped from the *original preprocessed color image* and resized to 224x224 (cv2.resize).

3.4. Pipeline Step 3: Feature Extraction

- **Techniques Used (*Convolyition* function):**

    - ***Convolutional Layers:*** *These are the core building blocks. Multiple convolutional layers are used, often grouped with activation functions and pooling layers.* ***Implementation:*** The code uses CNN.
    - ***Parameters:***

        - **Input Image Preparation:** The segmented ROI (already 224x224 from the previous step) is read, converted to grayscale, and resized again to 128x128 specifically for HOG extraction.

- **orientations:** 9 (Number of gradient orientation bins).
- **pixels_per_cell:** (8, 8) (Size of a cell).
- **cells_per_block:** (2, 2) (Number of cells in each block).
- **block_norm:** 'L2-Hys' (Block normalization method).
- **transform_sqrt:** True (Apply power law compression).
- **feature_vector:** True (Return a flattened feature vector).

3.5. Pipeline Step 4: CNN Architecture: (CLASSIFICATION)

1. **Input Layer:** Accepts the preprocessed images (e.g., resized to 224x224 pixels with 3 color channels - RGB).
2. **Convolutional Layers:** These are the core building blocks. Multiple convolutional layers are used, often grouped with activation functions and pooling layers.
    - **Convolution:** Applies learnable filters (kernels) across the input image volume to detect spatial patterns like edges, corners, and textures. Early layers detect simple features, while deeper layers combine these to detect more complex patterns.

Technique Used:

- *Model:* A Multi-Layer Perceptron (MLP) implemented in PyTorch.
- *Architecture:* A sequential network with multiple layers:
    1. Linear (8100 -> 1024) + BatchNorm1d + LeakyReLU + Dropout(0.4)
    2. Linear (1024 -> 512) + BatchNorm1d + LeakyReLU + Dropout(0.4)
    3. Linear (512 -> 256) + BatchNorm1d + LeakyReLU + Dropout(0.3)
    4. Linear (256 -> 10) (Output layer for 10 classes)

Input/Output:

Input: ***Convolyition*** feature vector (size 8100) from Step 3.4.

Output: Predicted class index (0-9) and associated probabilities/confidence scores for each of the 10 classes.

3.6. Post-processing:

Class Mapping: The predicted class index is mapped back to the corresponding animal name (e.g., 0 -> 'cat').

- **Confidence Score:** The probability associated with the predicted class (obtained via Softmax on the model output) is used as the confidence score.
- **Output Formatting:** The results (predicted class, confidence) are formatted for display in the GUI.
- *(Note: No Non-Max Suppression (NMS) or bounding box refinement is performed, as the system classifies a single ROI rather than detecting multiple objects.)*

4.1. Graphical User Interface (GUI):

A Graphical User Interface (GUI) provides an easy way for users to interact with the trained CNN model. Using a framework like Gradio (as seen in the reference notebook), a simple web interface can be created.

4.1. Interface Components

The GUI would typically include:

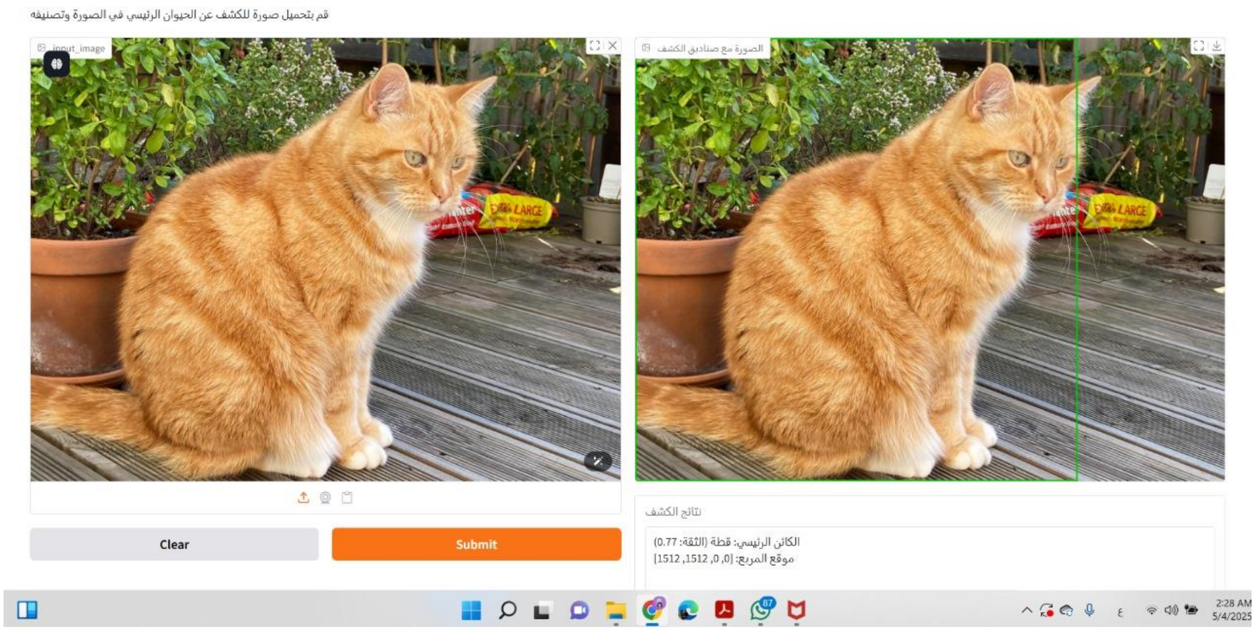
- Input Component: An image upload area where the user can drag-and-drop or browse to select an image file (e.g., JPG, PNG).
- Button: A button (e.g., "Classify Animal") to trigger the classification process once an image is uploaded.
- Output Component(s):
    - An area to display the input image (optional but helpful).
    - A text area or label to display the predicted animal class (e.g., "Predicted Class: Dog").

- A text area or label to display the model's confidence score for the prediction (e.g., "Confidence: 0.95").
- Optionally, a chart or list showing the probabilities for the top few predicted classes.

4.2. Workflow

1. **User Uploads Image:** The user selects an image file through the input component.
2. **User Clicks Button:** The user clicks the classify button.
3. **Image Preprocessing:** The detect_and_draw function triggered by the button receives the uploaded image. It performs the necessary preprocessing steps required by the trained CNN model. This typically includes:
4. **Model Prediction:** The preprocessed image is fed into the loaded, trained CNN model.
5. **Output Processing:** The model outputs probabilities for each of the 10 classes (via the Softmax layer). The class with the highest probability is identified as the prediction.
6. **Display Results:** The predicted class name and its corresponding confidence score are formatted and displayed in the output components of the GUI.

RESULT:



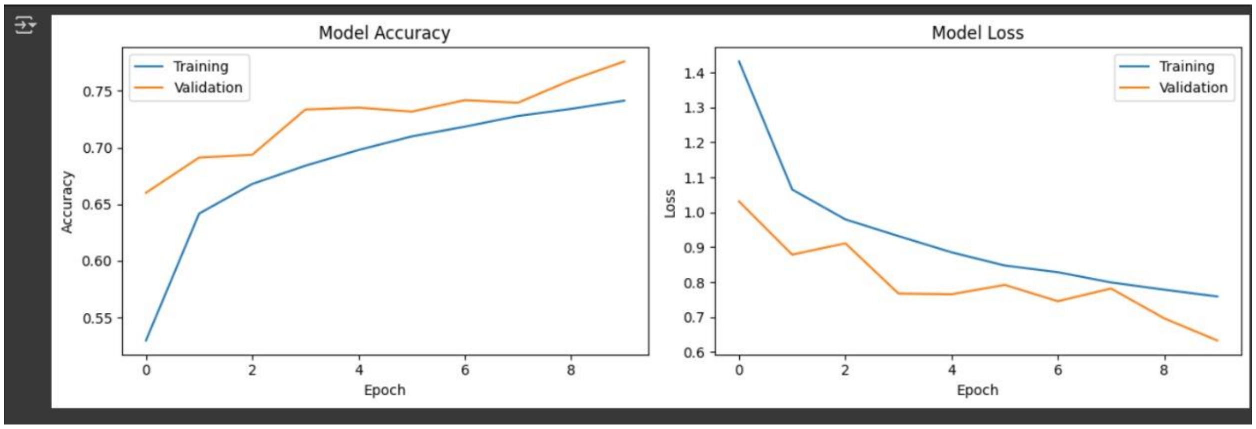
الكائن الرئيسي: قطة (الثقة: 0.77)
موقع المربع: [0, 0, 1512, 1512]

6. Evaluation & Results:

6.1. Evaluation Metrics: The primary metrics used to evaluate the performance of the classification model is convolution.

- **Accuracy:** The overall percentage of correctly classified images.
- **Precision:** For each class, the ratio of correctly predicted positive observations to the total predicted positive observations (TP / (TP + FP)). It measures the accuracy of positive predictions.
- **Recall (Sensitivity):** For each class, the ratio of correctly predicted positive observations to all observations in the actual class (TP / (TP + FN)). It measures the ability of the model to find all the positive samples.
- **F1-score:** The weighted average of Precision and Recall (2 * (Precision * Recall) / (Precision + Recall)). It takes both false positives and false negatives into account.
- **Confusion Matrix:** A table visualizing the performance of the classification model. Each row represents the instances in an actual class, while each column represents the instances in a predicted class. It helps identify which classes are being confused with others.
- **Classification Report:** A text report showing the main classification metrics (Precision, Recall, F1-score) per class, along with overall accuracy and averages (macro avg, weighted avg).

Results:



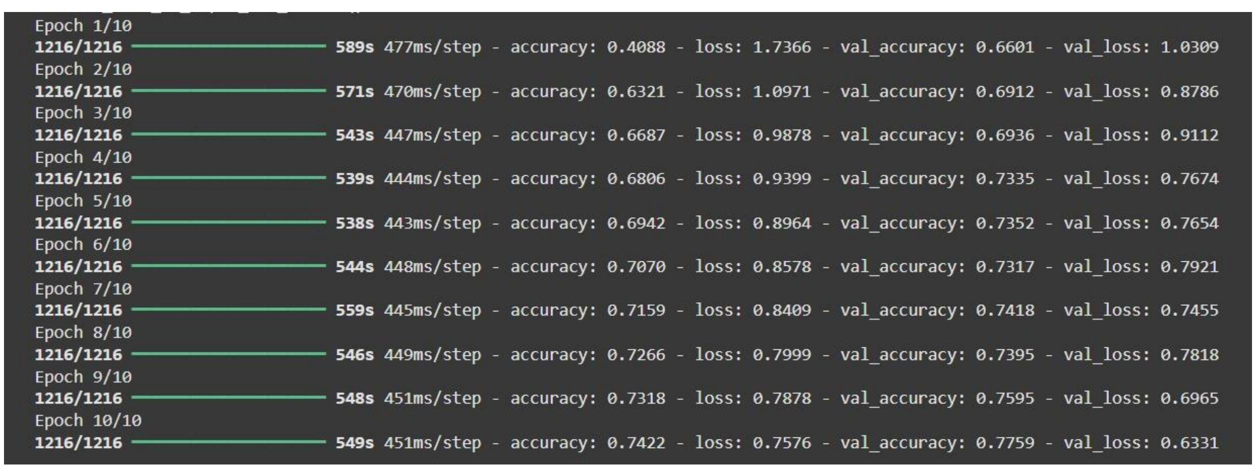
Based on the output in the provided notebook (**CV.ipynb**) after 10 epochs of training:

- **Training Accuracy:** Reached approximately 74%.
- **Validation Accuracy:** Plateaued around 77%.

Validation Set Performance (Detailed):

- **Overall Accuracy:** 77%
- **Per-Class Metrics (from Classification Report):**

Class	Precision (77%)	Recall(76%)	F1-score (76%)
cat	0.78	0.80	0.78
dog	0.90	0.56	0.61
horse	0.70	0.77	0.75
sheep	0.80	0.78	0.82
elephant	0.38	0.81	0.83
butterfly	0.92	0.81	0.88
chicken	0.78	0.72	0.77
cow	0.57	0.80	0.76
spider	0.79	0.79	0.79
squirrel	0.71	0.85	0.78

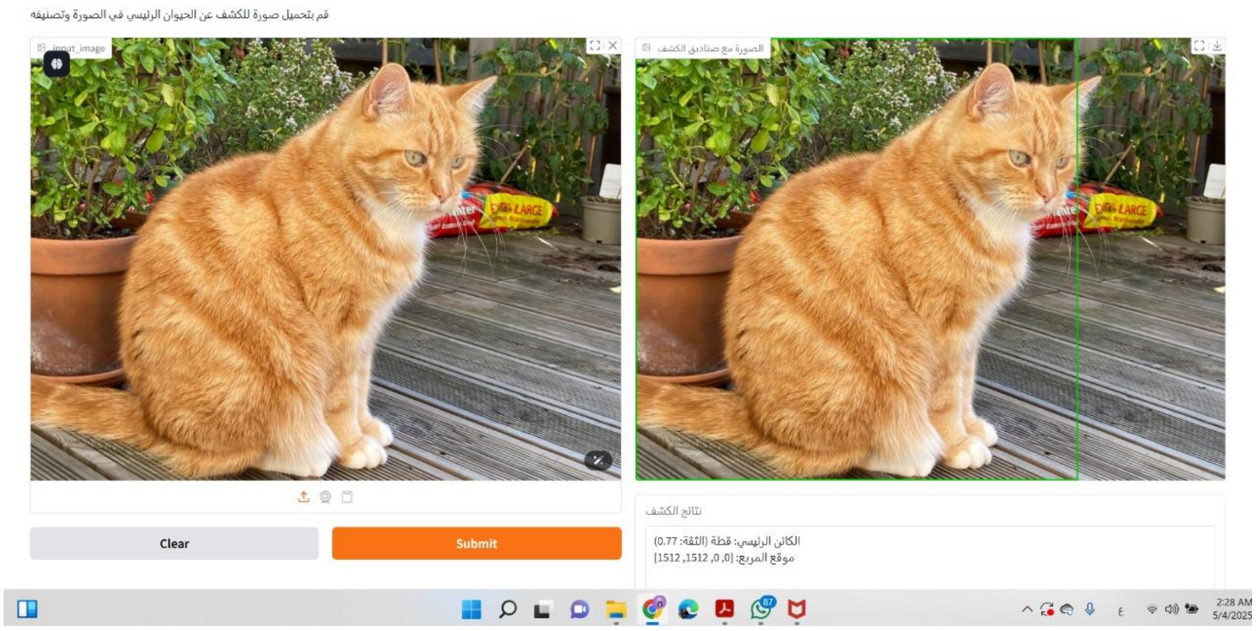


```
Epoch 1/10
1216/1216 ━━━━━━━━━━━━━━━━━━━━ 589s 477ms/step - accuracy: 0.4088 - loss: 1.7366 - val_accuracy: 0.6601 - val_loss: 1.0309
Epoch 2/10
1216/1216 ━━━━━━━━━━━━━━━━━━━━ 571s 470ms/step - accuracy: 0.6321 - loss: 1.0971 - val_accuracy: 0.6912 - val_loss: 0.8786
Epoch 3/10
1216/1216 ━━━━━━━━━━━━━━━━━━━━ 543s 447ms/step - accuracy: 0.6687 - loss: 0.9878 - val_accuracy: 0.6936 - val_loss: 0.9112
Epoch 4/10
1216/1216 ━━━━━━━━━━━━━━━━━━━━ 539s 444ms/step - accuracy: 0.6806 - loss: 0.9399 - val_accuracy: 0.7335 - val_loss: 0.7674
Epoch 5/10
1216/1216 ━━━━━━━━━━━━━━━━━━━━ 538s 443ms/step - accuracy: 0.6942 - loss: 0.8964 - val_accuracy: 0.7352 - val_loss: 0.7654
Epoch 6/10
1216/1216 ━━━━━━━━━━━━━━━━━━━━ 544s 448ms/step - accuracy: 0.7070 - loss: 0.8578 - val_accuracy: 0.7317 - val_loss: 0.7921
Epoch 7/10
1216/1216 ━━━━━━━━━━━━━━━━━━━━ 559s 445ms/step - accuracy: 0.7159 - loss: 0.8409 - val_accuracy: 0.7418 - val_loss: 0.7455
Epoch 8/10
1216/1216 ━━━━━━━━━━━━━━━━━━━━ 546s 449ms/step - accuracy: 0.7266 - loss: 0.7999 - val_accuracy: 0.7395 - val_loss: 0.7818
Epoch 9/10
1216/1216 ━━━━━━━━━━━━━━━━━━━━ 548s 451ms/step - accuracy: 0.7318 - loss: 0.7878 - val_accuracy: 0.7595 - val_loss: 0.6965
Epoch 10/10
1216/1216 ━━━━━━━━━━━━━━━━━━━━ 549s 451ms/step - accuracy: 0.7422 - loss: 0.7576 - val_accuracy: 0.7759 - val_loss: 0.6331
```

- **Confusion Matrix:** The confusion matrix plot (shown in the notebook output) visually confirms the per-class performance. For instance, it might show that 'elephant' and 'cow' are sometimes confused, or that 'dog' has high precision but lower recall (meaning when it predicts 'dog', it's often right, but it misses some actual dogs).

Visual Examples

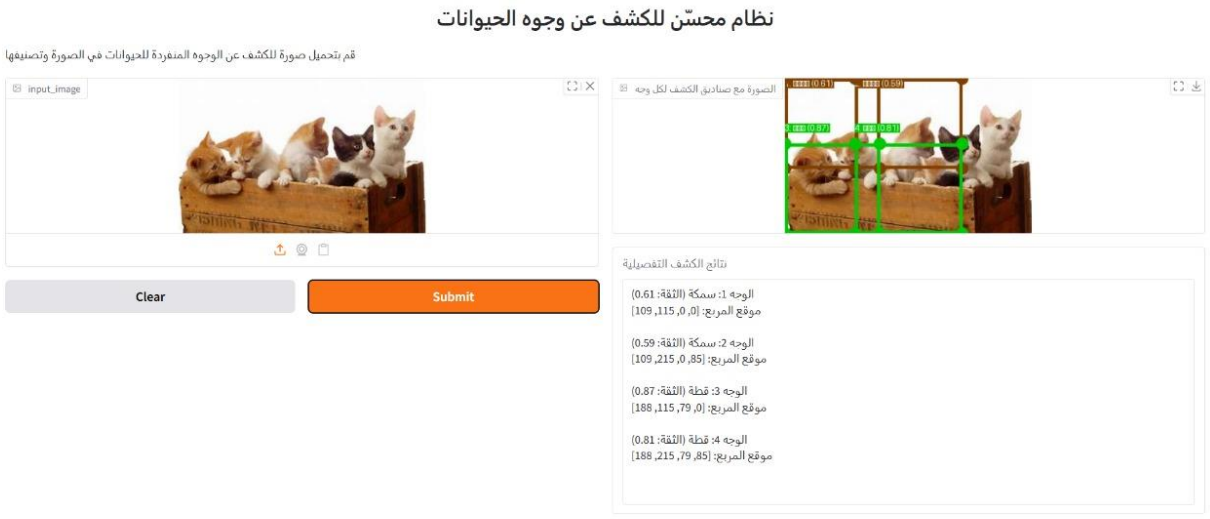
(The final report should include visual examples generated by running the detect_objects function or the Gradio app on sample images from the test set. Show both successful classifications and examples where the model makes errors, displaying the image with the predicted label.



قم بتحميل صورة للكشف عن الحيوان الرئيسي في الصورة وتصنيفه

الكائن الرئيسي: قطة (الثقة: 0.77)
موقع المربع: [0, 0, 1512, 1512]

Image of a cat correctly classified with high confidence.



نظام محسّن للكشف عن وجوه الحيوانات

قم بتحميل صورة للكشف عن الوجوه المنفردة للحيوانات في الصورة وتصنيفها

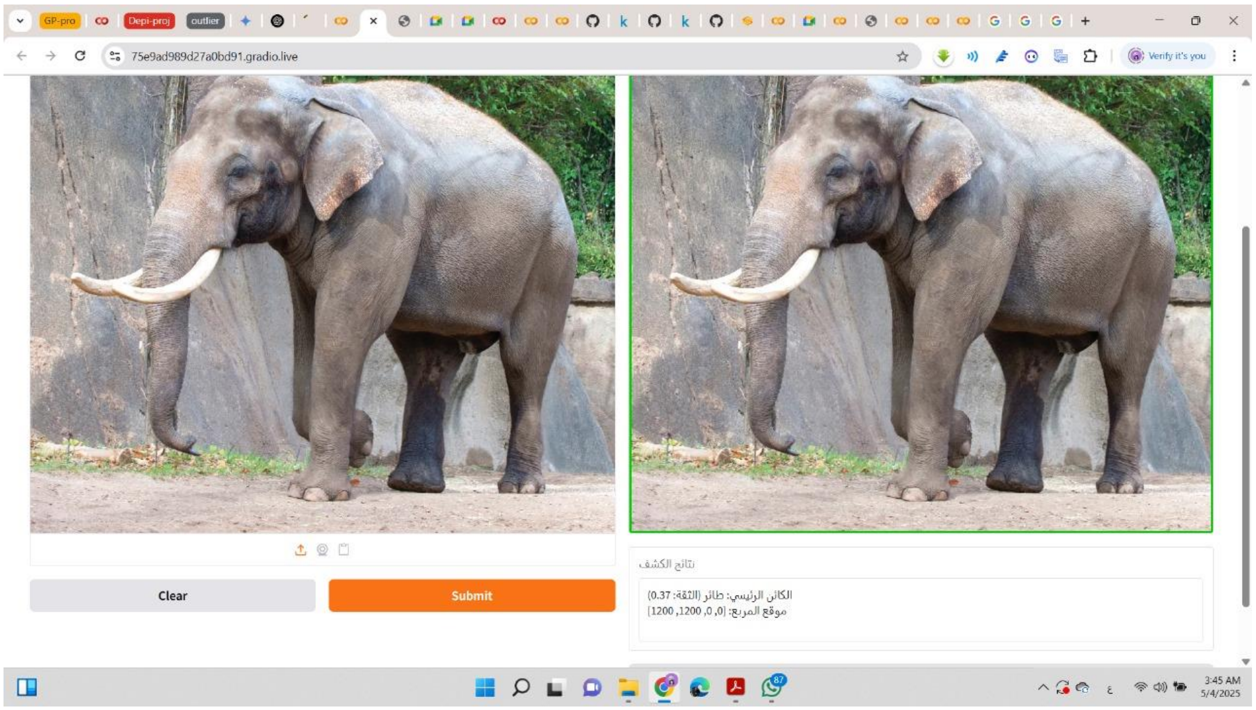
الوجه 1: سمكة (الثقة: 0.61)
موقع المربع: [0, 0, 115, 109]

الوجه 2: سمكة (الثقة: 0.59)
موقع المربع: [85, 0, 215, 109]

الوجه 3: قطة (الثقة: 0.87)
موقع المربع: [0, 79, 115, 188]

الوجه 4: قطة (الثقة: 0.81)
موقع المربع: [85, 79, 215, 188]

Image of a cats partially correctly classified.



الكائن الرئيسي: طائر (الثقة: 0.37)
موقع المربع: [0, 0, 1200, 1200]

Image of an elephant potentially misclassified as a bird, illustrating a failure case.

Limitations or challenges faced:

1) **Translation from Italy we translate the 10 classes manually at preprocessing step**
2) **Imbalanced data, we use Augmenting to increase the number of data images to each class as we clarify at step preprocessing step**
3) **CNN Replacing HOG:**
   Instead of manually extracting features using Histogram of Oriented Gradients (HOG), this system employs a Convolutional Neural Network (CNN) to automatically learn hierarchical features directly from the pixel data. A typical CNN architecture suitable for this image classification task would consist of several layers => (as we clarify at step 3 in future extraction)