

# Smart Web Art Gallery

Adriana Cristiana Bîrzeanu, Paul Stefan Popescu, Marian Cristian Mihaescu

University of Craiova, Craiova, Dolj, Romania  
*birzeanu.adriana.u3i@student.ucv.ro, stefan.popescu@edu.ucv.ro,*  
*crislian.mihaescu@edu.ucv.ro*

**Abstract.** Digital art galleries have become increasingly popular; the intersection of technology and art has opened new avenues for creative works. This paper introduces the Smart Web Art Gallery System, a novel platform designed to revolutionize online art curation, display, and user interaction. By leveraging advanced technologies such as artificial intelligence (AI), and data analytics, the system offers a dynamic and personalized user experience. A key component of the system is the integration of the YOLOv3 (You Only Look Once, version 3) model, a state-of-the-art real-time object detection algorithm. Implemented using Flask, a lightweight web framework, YOLOv3 enhances the system's ability to recognize and categorize artworks accurately. The overall application is developed using ASP.NET Core MVC, ensuring robust and scalable performance. Experiments demonstrate the system's effectiveness making it more inclusive and engaging for diverse audiences worldwide.

**Keywords:** Art gallery, YOLO, Tag detection

DOI: 10.37789/ijusi.2023.16.4.2

## 1. Introduction

In the contemporary digital age, the intersection of technology and art has opened new avenues for the appreciation and dissemination of creative works. While still vital, traditional art galleries are increasingly complemented by innovative digital platforms offering enhanced accessibility and interactive experiences. This paper introduces the concept and implementation of a Smart Web Art Gallery System, designed to revolutionize the way art is curated, displayed, and experienced online.

The Smart Web Art Gallery System leverages cutting-edge technologies such as artificial intelligence (AI) and data analytics to create a dynamic and personalized user experience. Unlike conventional online galleries that merely replicate physical exhibits, this system adapts to individual user

preferences, offering tailored recommendations and immersive interactions with artworks.

A significant technological component of this system is the integration of the YOLOv3 (You Only Look Once, version 3) model, an advanced real-time object detection algorithm. YOLOv3 is employed to enhance the system's capability in recognizing and categorizing artworks with high accuracy. This model analyzes visual content to automatically identify and classify different styles, periods, and features of artworks. By doing so, it enables more precise recommendations and interactions based on the detected attributes. The model is implemented using Flask, a lightweight WSGI web application framework, ensuring efficient and scalable deployment.

The application itself is developed using ASP.NET Core MVC, a robust framework for building scalable and high-performance web applications. This combination of Flask for the YOLOv3 model and ASP.NET Core MVC for the overall application infrastructure allows for seamless integration and efficient handling of complex tasks such as real-time object detection and dynamic content delivery.

Key features of the system include AI-driven art recommendations, which analyze user behavior and preferences to suggest relevant artworks, and AR functionalities that allow users to visualize art pieces within their own spaces. The YOLOv3 model plays a crucial role in this by providing a detailed analysis of each artwork, thus enriching the data used for personalization. Additionally, the system incorporates advanced data analytics to provide artists and curators with valuable insights into visitor engagement and trends, fostering a more informed and responsive approach to art curation.

By bridging the gap between traditional art exhibition methods and modern technological capabilities, the Smart Web Art Gallery System aims to democratize art access, making it more inclusive and engaging for diverse audiences worldwide. This paper will explore the development process, core functionalities, and potential impact of the system, highlighting its significance in the evolving landscape of digital art presentation.

## **2. Related Work**

The usage of deep learning algorithms for object detection in visual arts is a well-debated subject as presented in Bengamra et al. (2024). The authors state

that where many methods exist, a deep review of the literature concerning object detection in visual art is still lacking. After reviewing several related papers, this study presents a comprehensive review, including an overview of major computer vision applications for visual art, a presentation of previous related surveys, and a comprehensive overview of relevant object detection methods for artistic images. Considering the studied object detection methods, we propose a new taxonomy based on the supervision learning degree, the adopted framework, the adopted methodology (classical or deep-learning-based method), the type of object to detect, and the depictive style of the painting images. Then the several challenges for object detection in artistic images are described and the proposed ways of solving some encountered problems are discussed. In addition, available artwork datasets and metrics used for object detection performance evaluation are presented.

Another art form is analyzed in Bomfim et al. (2022) where the paper describes a deep learning-based methodology to accurately detect urban graffiti in complex images. The different graffiti varieties and the multiple variabilities present in these artistic elements on street scenes (such as partial occlusions or their reduced size) make this object detection problem challenging. Their experimental results using different datasets endorse the effectiveness of this proposal.

Regarding machine learning algorithms used in visual art there is a paper Falomir et al. (2018) that used k-Nearest Neighbor as in Pandey et al. (2017) and support vector machine presented by Peng et al. (2002) techniques for learning the features of paintings from the Baroque, Impressionism and Post-Impressionism styles. Specifically, two classifiers are built, and two different parameterizations have been applied for the QCD. For testing QArt-Learn approach, the Painting-91 dataset has been used, from which the paintings corresponding to Velázquez, Vermeer, Monet, Renoir, van Gogh and Gauguin were extracted, resulting in a set of 252 paintings. The results show that categorization accuracies are higher than 65%, comparable to accuracies obtained in the literature. However, QArt-Learn uses qualitative color names to describe style color palettes linguistically, so non-experts in art can better understand them since QCDs are aligned with human perception.

Regarding review and summarization papers about deep learning models applied on art, there is a paper published by Diaz et al. (2020) in which was carried out in the context of the Digital Humanities project ChIA, authors present an approach for testing different commercial (Clarifai, IBM Watson,

Microsoft Cognitive Services, Google Cloud Vision) and open-source (YOLO) computer vision (CV) tools on a set of selected cultural food images from the Europeana collection with regard to producing relevant concepts. The project aims to improve access to implicit cultural knowledge contained in images and increase analysis possibilities for scientific research, content providers, and educational purposes. Preliminary results showed that not only quantitative output results are important, but also the quality of concepts generated.

In his paper, Diaz et al. (2020) compare two influential deep learning algorithms in image processing and object detection, Mask R-CNN and YOLO. The authors state that it is also a challenging task to understand subtle details in their surroundings. For instance, radiance conditions, background clutter, and partial or full occlusion. When a machine tries to interact with human or try to take pictures, it becomes hard for them to magnify the details of a human surrounding. In this study, we have focused on detecting humans effectively.

The main focus of the paper published by Sumit et al. (2020) objective of the work is to compare the performance of YOLO and Mask R-CNN, which unveils the inability of Mask R-CNN to detect tiny human figures among other prominent human images, and illustrate YOLO was successful in detecting most of the human figures in an image with higher accuracy. Therefore, the paper evaluates and differentiates the performance of YOLO from the deep learning method Mask R-CNN in two points, detection ability and computation time. Since machine learning algorithms are mostly data-specific, the authors believe that the presented results might vary with the varying nature of the data under observation. In another way, the presented data might be seen as a counter-example of unveiling the detection inaccuracy of the Mask R-CNN.

### **3. Proposed approach**

#### **3.1 System architecture**

The provided diagram (Figure 1) illustrates a comprehensive system architecture composed of various components that collaborate to deliver functionality to users. The system can be divided into three primary layers:

Frontend, Backend, and External Services.

**The front end** is responsible for the user interface and user experience. It is implemented using HTML, CSS, and Bootstrap for styling, along with JavaScript and jQuery for dynamic content and interactivity. Users interact with the system through this layer, uploading images, inputting data, and viewing results. The use of these technologies ensures a responsive and user-friendly interface.

**The backend** handles the application logic, processes user inputs from the front end, and interacts with the database and external services. The backend is built using ASP.NET Core MVC, which handles HTTP requests and renders views, providing a robust framework for developing web applications. Entity Framework Core is utilized for database interactions, facilitating CRUD (Create, Read, Update, Delete) operations and database management, ensuring seamless integration with the SQL Server database. ASP.NET Identity is employed for authentication and authorization, managing user accounts and roles, which enhances the security of the application.

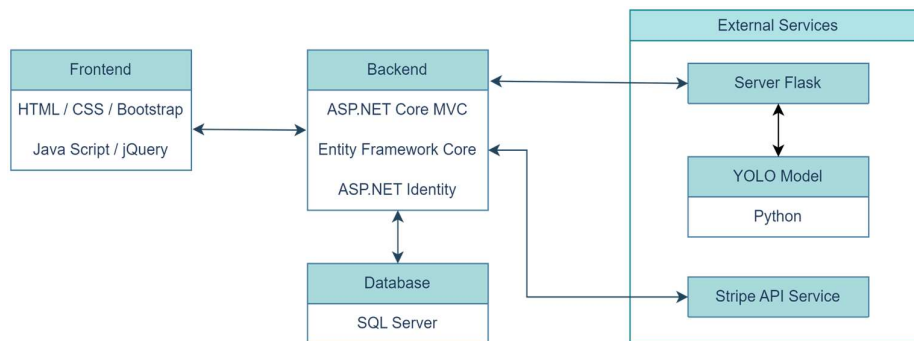


Figure 1. System Architecture Description

The **Database**, implemented using SQL Server, stores persistent data including user information, uploaded images, and tags generated by the YOLO model. The backend interacts with the database through Entity Framework Core to perform various operations, ensuring data integrity and consistency.

The **External Services** component consists of a Flask server, the YOLO model, and the Stripe API service. The Flask server, a lightweight Python web server, hosts the YOLO model for object detection which is more

detailed by Diwan et al. (2023). The backend communicates with this server to send images for processing. The YOLO model which was also described by Stancel et al (2019), implemented in Python, performs object detection on the uploaded images, processes them, and returns detected objects as tags. Additionally, the Stripe API service is used for payment processing; the backend interacts with this service to handle financial transactions securely.

The workflow begins when users upload images via the front end. The backend receives the image and sends it to the Flask server. The Flask server then uses the YOLO model to detect objects in the image. The detected objects are returned to the backend as tags. The backend stores these tags in the SQL Server database and displays them to the user via the front end. If required, the backend interacts with the Stripe API service to process payments.

This architecture ensures a clear separation of concerns, with each layer and component handling distinct responsibilities. The front end provides an interactive user interface, the back end manages application logic and data flow, and external services handle specialized tasks like object detection and payment processing. This modular approach enhances maintainability, scalability, and the overall robustness of the system. By leveraging the strengths of each technology and component, the system delivers a seamless and efficient user experience.

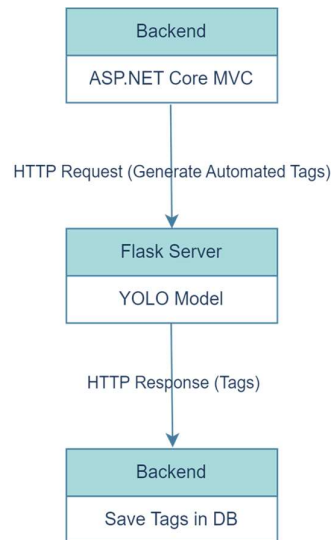


Figure 2. Tag Generation Workflow

The diagram provided (Figure 2) outlines the workflow for generating automated tags using a YOLO model through a Flask server, integrated with an ASP.NET Core MVC backend.

The process begins in the **Backend**, implemented using ASP.NET Core MVC. When a user requests to generate automated tags for an uploaded image, the backend sends an HTTP request to the Flask server. This request includes the image data to be processed.

Upon receiving the request, the **Flask Server** utilizes the **YOLO Model** to perform object detection on the image. The YOLO model analyzes the image and identifies objects within it, generating corresponding tags for each detected object.

After processing the image, the Flask server sends an HTTP response back to the backend. This response contains the generated tags.

Finally, the backend receives the tags and saves them into the database. This completes the tag generation process, allowing the tags to be used for further application features or displayed to the user. This workflow ensures an efficient and automated way of tagging images, leveraging advanced object detection capabilities provided by the YOLO model.

### 3.2 Pipeline Breakdown into Modules

The system for generating automated tags is composed of several interconnected modules, each playing a crucial role in ensuring the functionality and efficiency of the overall process.

The front end serves as the user interface, developed using HTML, CSS, Bootstrap, JavaScript, and jQuery. This module is responsible for capturing user inputs, including image uploads, and providing a responsive and intuitive interface. Users interact with the front end to initiate the tag generation process by uploading images and requesting automated tags.

The Backend module, built with ASP.NET Core MVC, acts as the intermediary between the front end and the external services. It handles HTTP requests from the front end, processes the data, and communicates with external services. The backend uses Entity Framework Core for database operations and ASP.NET Identity for user authentication and management. Upon receiving an image upload request, the backend sends an HTTP request to the Flask server for tag generation.

The Flask Server, written in Python, hosts the YOLO Model for object detection. When the backend sends an image for processing, the Flask server utilizes the YOLO model to analyze the image and identify objects. The YOLO model, known for its speed and accuracy, processes the image and generates corresponding tags, which are then sent back to the backend as an HTTP response.

The Database, implemented using SQL Server, stores the generated tags along with other related data. The backend saves the tags received from the Flask server into the database, ensuring that they are readily available for future retrieval and use within the application.

Additionally, the system integrates the **Stripe API Service** for handling payments. This external service allows users to make secure transactions, adding a layer of functionality for applications requiring payment processing.

This modular architecture ensures a clean separation of concerns, with each module focusing on a specific aspect of the system. The frontend handles user interactions, the backend manages data flow and business logic, the Flask server processes images and generates tags using the YOLO model, and the database securely stores the data. This design enhances maintainability, scalability, and ease of development, allowing each module to be developed and optimized independently.



The use case diagram (Figure 3) depicts the interactions between three types of users and the functionalities of an online art platform. The three user types are general users, registered users, and administrators, each represented by distinct stick figures and arrows in different colors.

General users, indicated by the green arrows, have access to the following functionalities: viewing products, searching for products, registering on the platform, and viewing the art gallery which was also addressed by Chou et al. (2019). These actions do not require logging into the system and are accessible to anyone visiting the website.

Registered users, represented by the orange arrows, gain access to additional features after logging in. They can view their user cart, use a wish list, add products, edit their profile, view their order history, and add comments to artworks. Moreover, they have access to Stripe payment integration for purchasing products and benefit from automated tag generation to enhance their browsing experience.

Administrators, shown with red arrows, possess advanced privileges. After logging in, they can edit the order stage, delete artworks, delete comments, and view the dashboard. These functionalities are critical for maintaining the platform's content and ensuring smooth operational management.

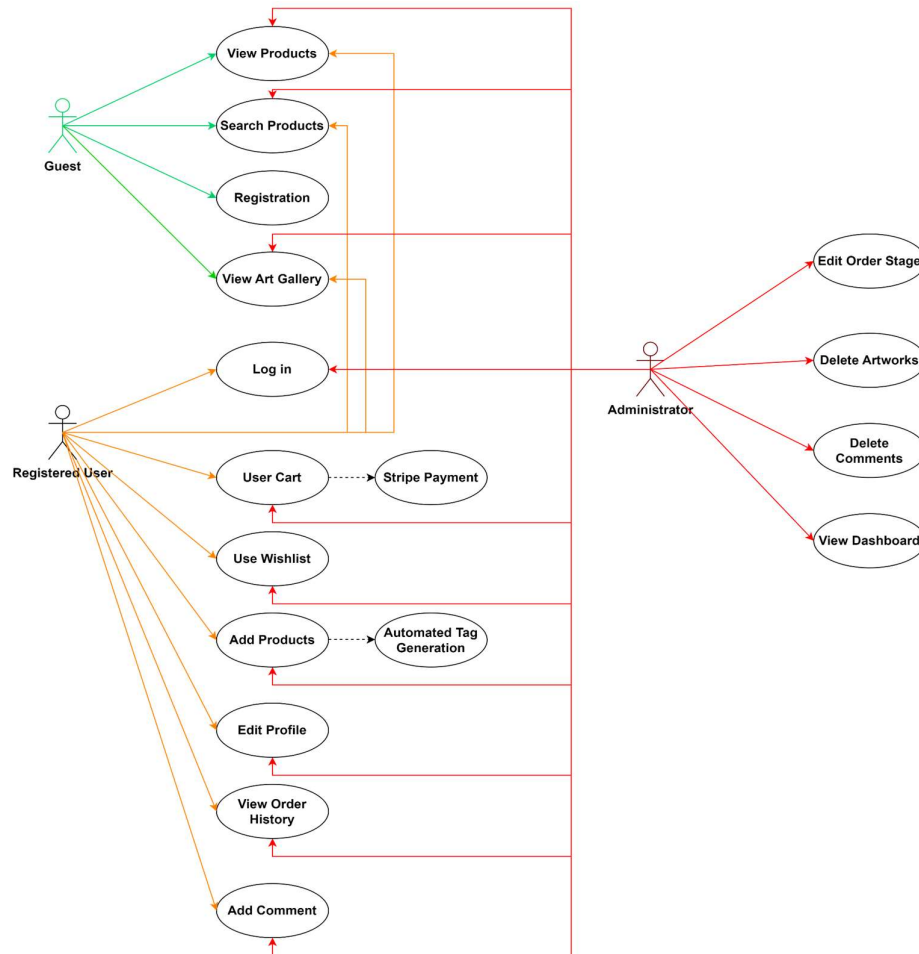


Figure 3. Use-Case Diagram

In summary, the diagram efficiently categorizes functionalities based on user roles, highlighting the accessible features for general users, the extended capabilities for registered users, and the administrative control available to administrators. This structured access ensures a streamlined user experience while maintaining secure and effective platform management.

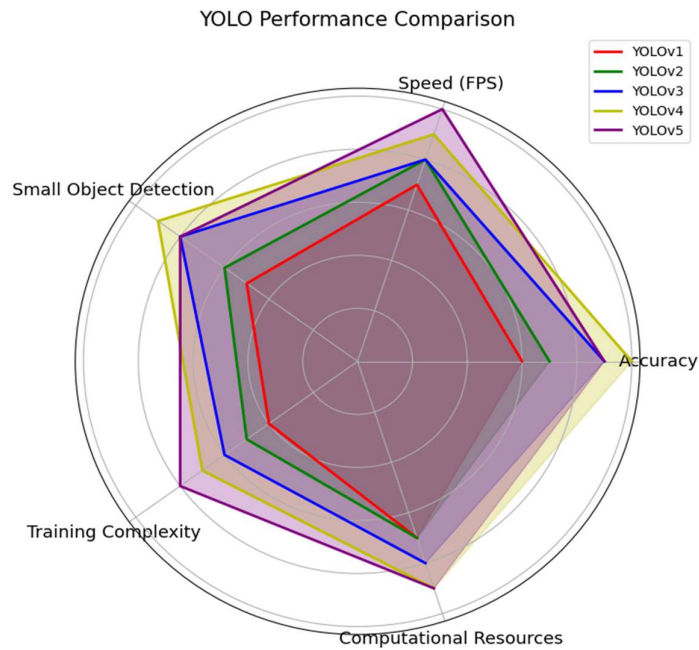


Figure 4. YOLO Performance Comparison

### 3.3 Image tagging functionality

The selection of YOLOv3 (You Only Look Once, version 3) for the image tagging functionality is driven by several key factors that align with the needs of our system. Here is an in-depth discussion of why YOLOv3 is the optimal choice for our application (Figure 4):

#### Balanced Accuracy

YOLOv3 provides a well-rounded balance between accuracy and speed. Compared to its predecessors, YOLOv1 and YOLOv2, version 3 has shown significant improvements in precision. This enhancement ensures that our system can accurately detect and tag objects within images, reducing the likelihood of errors and improving the overall reliability of the tag generation process. The balanced accuracy is crucial for maintaining a high standard of performance while still being efficient.

### **Real-Time Performance**

One of the standout features of YOLOv3 is its capability to process images in real-time. With a processing speed of approximately 20 frames per second (FPS) on a high-end GPU, YOLOv3 is well-suited for applications that require immediate results. This real-time performance is particularly beneficial for our system as it allows users to receive instant feedback on their uploaded images. This responsiveness enhances the user experience by minimizing waiting times and providing immediate insights.

### **Small-Object Detection**

YOLOv3 includes improvements specifically designed for the detection of small objects. In many practical scenarios, the ability to accurately identify and tag small objects within an image is critical. Whether it's for applications in surveillance, healthcare, or retail, the detection of small objects ensures that no important details are missed. YOLOv3's enhanced algorithms for small object detection make it a robust choice for comprehensive image analysis.

### **Resource and Performance Balance**

Another significant advantage of YOLOv3 is its efficient use of computational resources. While delivering high performance and accuracy, YOLOv3 maintains reasonable resource requirements, making it accessible for various applications. This balance is crucial for our system as it ensures that we can deploy the model without the need for excessively powerful hardware, thus reducing costs and making the technology more accessible.

### **Technical Improvements in YOLOv3**

YOLOv3 introduces several technical improvements over its predecessors:

- **Multi-Scale Predictions:** YOLOv3 predicts bounding boxes at different scales, which improves its ability to detect objects of varying sizes within the same image.
- **Feature Pyramid Network (FPN):** This allows YOLOv3 to use different layers of the network to detect small, medium, and large objects, enhancing its versatility.
- **Darknet-53 Backbone:** YOLOv3 uses the Darknet-53 architecture, which is a more powerful and efficient feature extractor than the ones used in

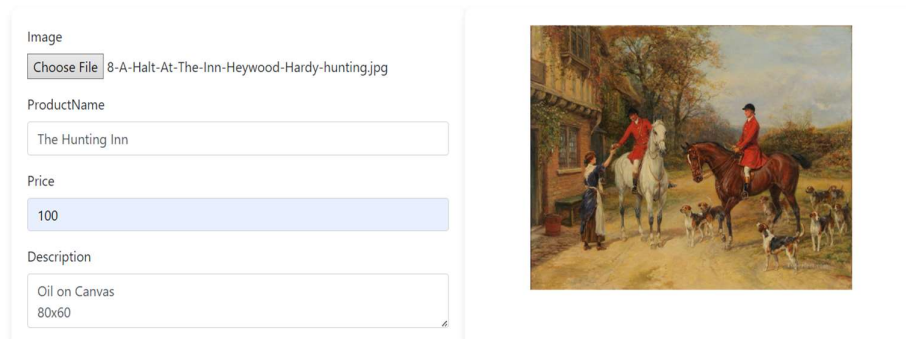
YOLOv1 and YOLOv2.

## 4. RESULTS

The tag generation functionality is a core feature of the system, enabling automated tagging of images using advanced object detection techniques. This process involves several key steps, each handled by different modules of the system to ensure seamless and efficient operation.

### 4.2 User Interaction and Image Upload

The process begins at the Frontend where users interact with the application. Built with HTML, CSS, Bootstrap, JavaScript, and jQuery, the frontend provides an intuitive interface for users to upload images. Users can select an image file, which is then displayed in a preview section (Figure 5). Once the image is uploaded, the user can request the generation of automated tags by clicking a designated button.



The screenshot shows a web form for uploading an image. On the left, there are four input fields: 'Image' with a 'Choose File' button and the filename '8-A-Halt-At-The-Inn-Heywood-Hardy-hunting.jpg'; 'ProductName' with the text 'The Hunting Inn'; 'Price' with the value '100'; and 'Description' with the text 'Oil on Canvas' and '80x60'. On the right, there is a preview of the image, which is a painting of a hunting scene with two riders on horses and several dogs.

Figure 5. Uploading images

### 4.2 Backend Request Handling

Upon user request, the Backend module, developed with ASP.NET Core MVC, takes over. The backend receives the image file from the frontend and prepares it for processing. This involves creating an HTTP request that includes the image file as part of the form data. The backend sends this

request to the Flask server, which hosts the object detection model.

### 4.3 Flask Server and YOLO Model Processing

The Flask Server is a crucial component, implemented in Python, responsible for running the YOLO (You Only Look Once) object detection model. When the server receives the image from the backend, it processes the image using the YOLO model. YOLO is renowned for its ability to detect multiple objects within an image quickly and accurately. It scans the image, identifies objects, and generates corresponding tags based on the detected objects. The YOLO model returns a list of tags, each representing an object found in the image (Figure 6).

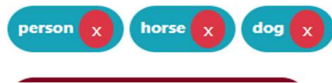


Figure 6. Tags found

### 4.4 Returning Tags to Backend

After the YOLO model processes the image and generates the tags, the Flask server sends an HTTP response back to the backend. This response includes the list of generated tags. The backend parses this response and extracts the tags.

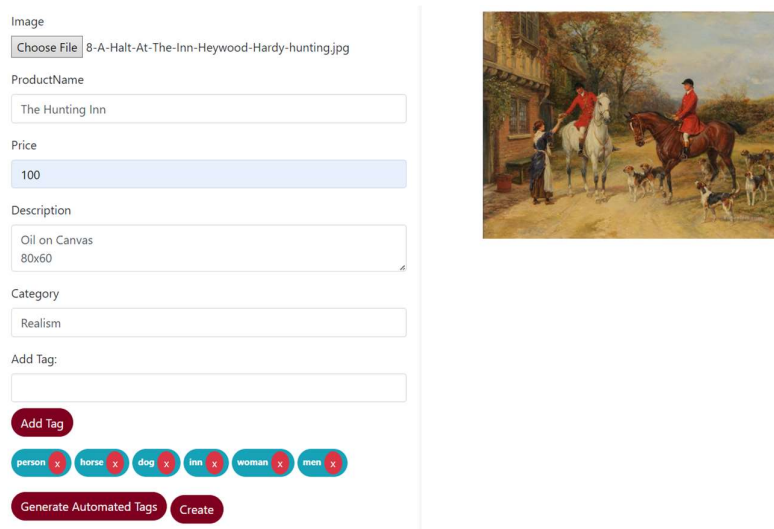
### 4.5 Storing Tags in Database

Once the backend receives the tags from the Flask server, it stores them in the Database. The database managed using SQL Server and Entity Framework Core, maintains a record of the generated tags along with other relevant data. This ensures that the tags are persistently stored and can be retrieved for future use.

### 4.6 Displaying Tags to User

Finally, the backend sends the generated tags back to the front end. The front end then displays these tags to the user (Figure 7), providing immediate feedback on the objects detected within the uploaded image. Users can view

the tags and, if necessary, delete or add new tags before finalizing their submission.



The image shows a web form for submitting an artwork. On the left is the form, and on the right is a preview of the artwork. The form fields are: 'Image' with a file selection button and the filename '8-A-Halt-At-The-Inn-Heywood-Hardy-hunting.jpg'; 'ProductName' with the text 'The Hunting Inn'; 'Price' with the value '100'; 'Description' with the text 'Oil on Canvas 80x60'; 'Category' with the text 'Realism'; and 'Add Tag:' with an empty input field. Below the input field is an 'Add Tag' button and a row of six tags: 'person', 'horse', 'dog', 'inn', 'woman', and 'men', each with a red 'x' icon to its right. At the bottom of the form are two buttons: 'Generate Automated Tags' and 'Create'. The preview image on the right is a painting titled 'Halt-At-The-Inn' by Heywood Hardy, depicting a scene with people on horseback and dogs in a rural setting.

Figure 5. Displaying Tags

## 5. CONCLUSION

The Smart Web Art Gallery System presents a significant advancement in the fusion of technology and art, demonstrating how digital platforms can enhance the accessibility and appreciation of creative works. Through the integration of the YOLOv3 model, implemented with Flask and supported by ASP.NET Core MVC, the system effectively automates the recognition and categorization of artworks, providing users with accurate and personalized art recommendations. Key findings from this research include the system's ability to process and identify various art styles and elements in real time, offering an interactive and engaging user experience.

The proposed solution bridges the gap between traditional art exhibition methods and modern technological capabilities, it also highlights areas for future research. In conclusion, the Smart Web Art Gallery System exemplifies how AI and modern web technologies can transform the traditional art gallery experience, making art more inclusive and engaging for diverse audiences worldwide. This research underscores the transformative

potential of digital platforms in the art world, paving the way for innovative approaches to art curation and presentation.

## References

- Bengamra, S., Mzoughi, O., Bigand, A., & Zagrouba, E. (2024). A comprehensive survey on object detection in Visual Art: taxonomy and challenge. *Multimedia Tools and Applications*, 83(5), 14637-14670.
- Bomfim, T. S., Nunes, É. D. O., & Sánchez, Á. (2022). Art Graffiti Detection in Urban Images Using Deep Learning. *ICT Applications for Smart Cities*, 1-20). Springer.
- Falomir, Z., Museros, L., Sanz, I., & Gonzalez-Abril, L. (2018). Categorizing paintings in art styles based on qualitative color descriptors, quantitative global features and machine learning (QArt-Learn). *Expert Systems with Applications*, 97, 83-94.
- Pandey, A., & Jain, A. (2017). Comparative analysis of KNN algorithm using various normalization techniques. *International Journal of Computer Network and Information Security*, 10(11), 36.
- Peng, D., Lee, F. C., & Boroyevich, D. (2002, June). A novel SVM algorithm for multilevel three-phase converters. In *2002 IEEE 33rd Annual IEEE Power Electronics Specialists Conference*. Proceedings (Cat. No. 02CH37289) (Vol. 2, pp. 509-513). IEEE.
- Díaz, J. L. P., Dorn, A., Koch, G., & Abgaz, Y. (2020, September). A comparative approach between different computer vision tools, including commercial and open-source, for improving cultural image access and analysis. In *2020 10th International Conference on Advanced Computer Information Technologies (ACIT)* 815-819. IEEE.
- Sumit, S. S., Watada, J., Roy, A., & Rambli, D. R. A. (2020, April). In object detection deep learning methods, YOLO shows supremum to Mask R-CNN. *Journal of Physics: Conference Series* (Vol. 1529, No. 4, p. 042086). IOP Publishing.
- Diwan, T., Anirudh, G., & Tembhrne, J. V. (2023). Object detection using YOLO: Challenges, architectural successors, datasets and applications. *Multimedia Tools and Applications*, 82(6), 9243-9275.
- Štancel, M., & Hulič, M. (2019). An introduction to image classification and object detection using YOLO detector. In *CEUR Workshop Proceedings* (Vol. 2403, pp. 1-8).
- Chou, Y. S., Wang, C. Y., Chen, M. C., Lin, S. D., & Liao, H. Y. M. (2019). Dynamic gallery for real-time multi-target multi-camera tracking. *16th IEEE International Conference on Advanced Video and Signal-Based Surveillance (AVSS)*, 1-8. IEEE.