EVOLUTION IN ELECTRICAL AND ELECTRONIC ENGINEERING
e-ISSN: 2756-8458

EEEE

Vol. 0 No. 0 (YEAR) 1-6
https://publisher.uthm.edu.my/periodicals/index.php/eeee

Detection of Foul Events in Football Using Deep Learning

Danial Aiman Bin Noor Azman[1], Dr. Mohamad Hairol Bin Jabbar[2]

[1] *Faculty of Electrical and Electronic Engineering/Universiti Tun Hussein Onn Malaysia , Parit Raja, 86400 Batu Pahat, Johor, Malaysia*
[2] *Faculty of Electrical and Electronic Engineering/Universiti Tun Hussein Onn Malaysia , Parit Raja, 86400 Batu Pahat, Johor, Malaysia*

*Corresponding Author: ce210158@student.uthm.edu.my.
DOI: https://doi.org/10.30880/eeee.2024.00.00.000

Article Info

Received: Day Month Year
Accepted: Day Month Year
Available online: Day Month Year

Keywords

Foul detection, football analytics, YOLOv8, deep learning, sports technology, automated refereeing.

Abstract

The detection of foul events in football is critical to ensuring fair play and improving refereeing accuracy. This study presents an automated system leveraging deep learning techniques, specifically the YOLOv8 object detection model, to identify tripping fouls in football matches. The dataset comprised 4,078 annotated images, mainly sourced from Premier League matches. Key results include a precision of 71.1%, recall of 81.8%, F1-score of 76.1%, and a mean average precision (mAP@50) of approximately 70%. While challenges such as dataset imbalance and background misclassification were noted, the system demonstrates promising accuracy and computational efficiency. Future work involves expanding detection to other foul types, integrating multi-camera perspectives, and optimizing for real-time applications. This study contributes significantly to sports analytics, enabling better decision-making in football officiating.

1. Introduction

Football is one of the most popular sports in the world [1]. It captivates millions of fans globally and generates economic, cultural and social impact. In modern football, there are many controversies and debates, especially regarding the accuracy of refereeing decisions. A technology system was introduced to assist the referee which is a Video Assistant Referee (VAR) was a significant step toward addressing these issues, by providing referees with video replays to review and make more informed decisions. VAR provides referees with the ability to review and make more informed decisions on incidents such as goals, penalties and red cards. VAR aims to reduce human error and enhance the accuracy of the sport.

However, while VAR has improved the decision-making process it is without its limitations and ultimately depends on human judgment to review incidents. VAR is intended to reduce refereeing errors and make football fairer, but its implementation is flawed. The subjectivity in determining what is "clear and obvious" leads to inconsistency and increased focus on refereeing decisions, making football arguably more unfair[2]. Sometimes judgment from referees can lead to errors or missed fouls. These limitations highlight the need for further advancement in technology to support referees more effectively. The a need for an advanced, automated system that can assist in detecting fouls more accurately and efficiently, thereby the fairness of the game..

2. Problem Statement

The current landscape of football referees uses video assistant referees (VARs) to review decisions when needed. The VAR used in an official match reviews decisions made by the field referee by using video replays. The referee can check the video replay on a monitor before making a final decision [3]. The real fouls sometimes go unnoticed and are not checked by the VAR system. The existing VAR system only assists the referees but the referee always has the final decision to accept or reject the VAR's advice.

This creates a need for an automated system that can accurately detect fouls and alert referees to ensure all incidents are reviewed. Many incidents in football matches lack efficient and accurate methods for detecting fouls in a football match. Sometimes, the VAR does not trigger the field referee to check the replay to analyze the fouls. Thus, there is an urgent need for an automated and reliable system capable of accurately detecting fouls in real-time or recorded footage.

3. Methodology

Methodology used in the project, the operational components, the design of the tools, and the flowcharts related to both project development and programming.

3.1 Hardware and Software Component

The project utilizes various hardware, software, and libraries to develop and implement a foul event detection system focusing on tripping fouls in football. Here's an overview of the components listed in Table 1:

Table 1 : Hardware and software

No	Hardware/Software/ Libraries	Description
1	Laptop	The laptop for developing, running, and managing the foul event detection system and its components.
2	Python Language	Python is the programming language of choice for implementing the foul event detection system.
3	Visual Studio Code	A code editor with features like debugging, syntax highlighting, and code refactoring for Python development.
4	Google Collab	A cloud platform with GPU/TPU support for running Python code and training deep learning models like YOLOv8.
5	Ultralytics YoloV8	A high-performance real-time object detection and image segmentation mode.
6	Open Source Computer Vision Library (CV)	An open-source library for image processing and computer vision tasks, supporting various programming languages.

3.2 System Workflow

The workflow begins with the Dataset Collection stage, where football match footage is collected, and tripping foul events are annotated using the Roboflow platform [4]. This process involves labeling specific incidents and preparing the data for training. Preprocessing steps, such as resizing images to 640×640 pixels, are carried out, and data augmentation techniques, including horizontal flipping, brightness adjustments, and rotations, are applied to enhance the dataset's diversity and reliability [5]. Once the dataset is prepared, it proceeds to the Model Training stage. Here, the preprocessed data is used to train the YOLOv8 object detection model on Google Colab, leveraging GPU acceleration for faster and more efficient training [6]. Parameters such as batch size, the number of epochs, and image resolution are configured to optimize the model for accurate foul detection. Finally, in the Deployment stage, the trained model is used to analyze video footage in real time through OpenCV [8]. Debugging and post-processing are performed using Visual Studio Code to ensure the system operates robustly and accurately under different scenarios [7].

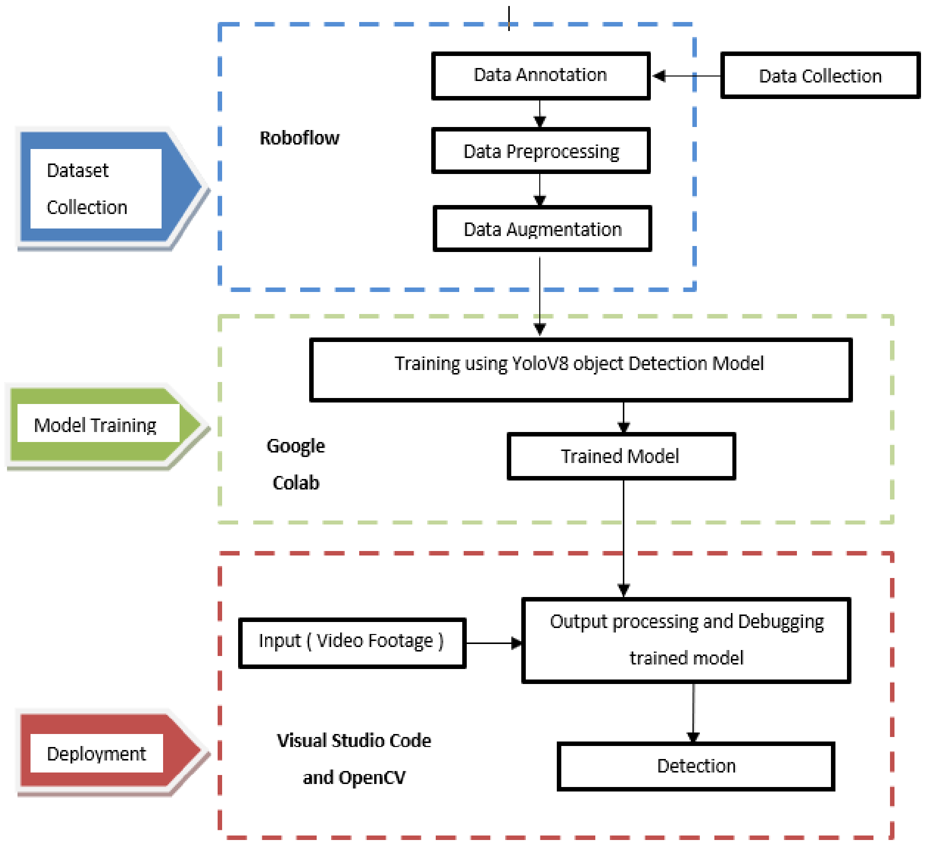


Figure 1: Block Diagram of the Foul Detection System.

Additionally, the development process follows the structured flow outlined in Figure 2. It starts with a literature review on foul event detection and progresses through dataset collection and model training. If the trained model meets the required accuracy, it is deployed for foul detection. Otherwise, the model undergoes further refinement to improve its performance.

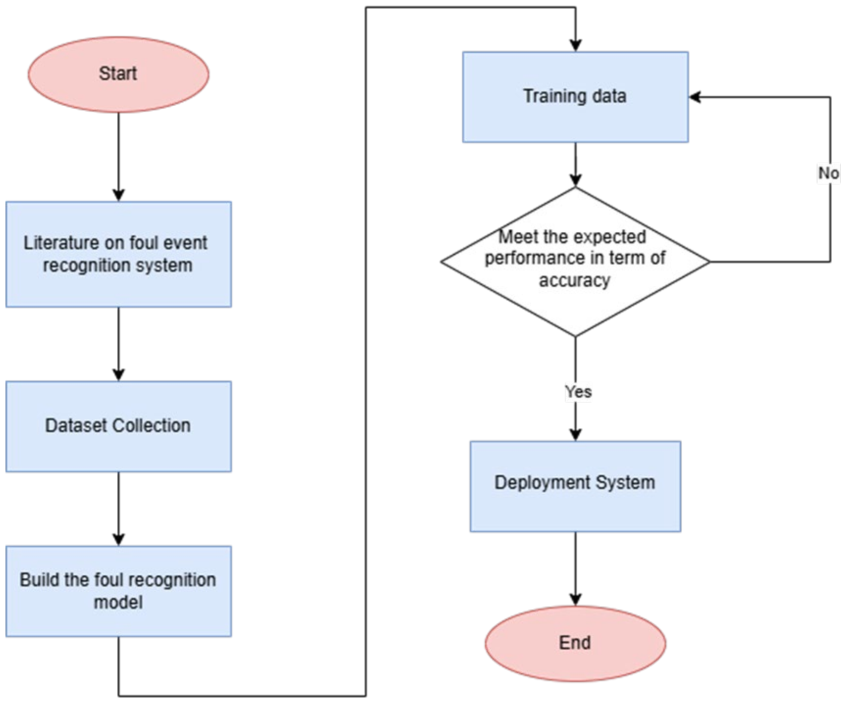

Start
Literature on foul event recognition system
Dataset Collection
Build the foul recognition model
Training data
Meet the expected performance in term of accuracy
No
Yes
Deployment System
End

Figure 2: Project Development Flowchart

4. Result and Discussion

The design of the system from data collection, annotation, and model training. It elaborates on how hardware components and software tools are integrated to facilitate the development and deployment of the system. The flow of data through various stages from data collection and annotation to model training and deployment is illustrated to highlight the operational workflow of the system.

4.1 Data Collection

A dataset of 4,078 annotated images was created using Premier League footage. Tripping fouls were labeled with bounding boxes using Roboflow [4]. The dataset was split into training (92%), validation (5%), and testing (3%) subsets. Preprocessing steps included resizing images to 640×640 pixels and applying augmentations like horizontal flipping, brightness adjustments, and rotations [5].

4.2 Model Training

The YOLOv8 model was trained on Google Colab, utilizing GPU acceleration for efficient processing [6]. Training parameters included a batch size of 16, 50 epochs, and an input image size of 640×640 pixels. Post-training, the model was deployed to analyze video footage using tools like OpenCV and Visual Studio Code for debugging [7], [8].

4.3 Trained Model Analysis

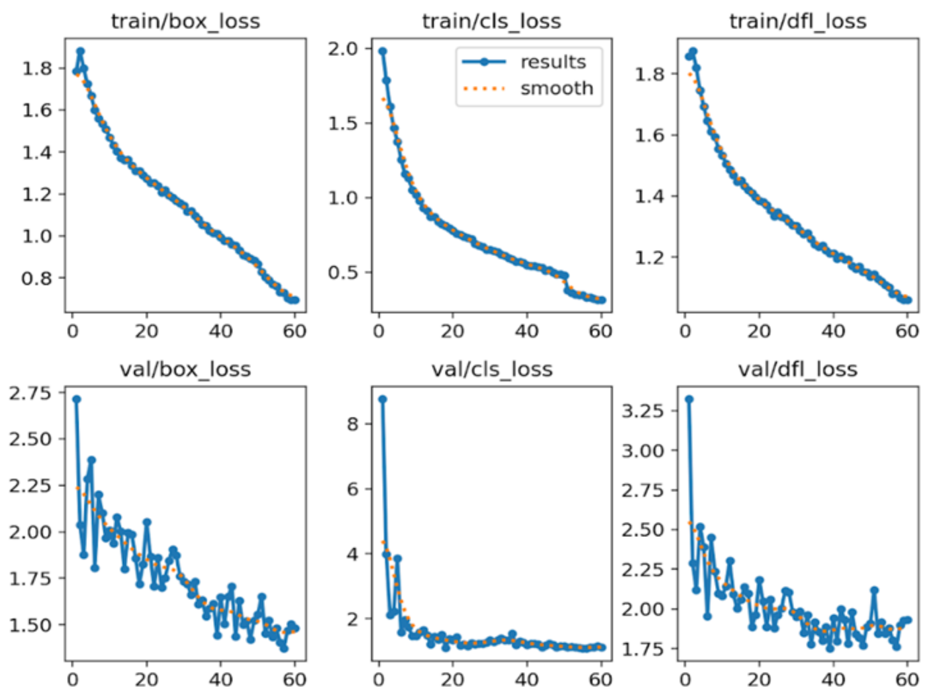
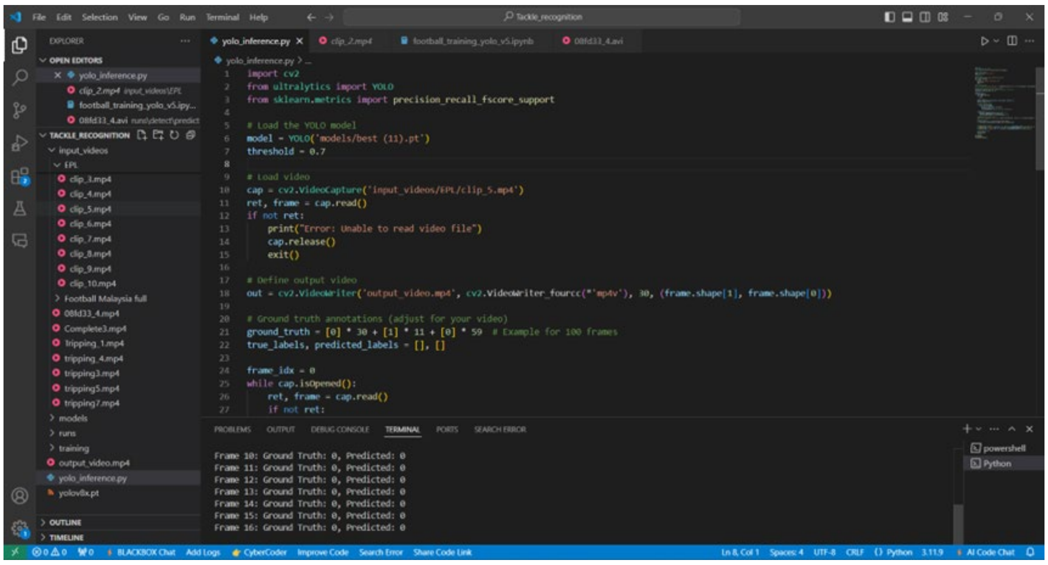


Figure 3: Training result

Figure 3 presents a summary of the training and validation metrics for a model that underwent training for 60 epochs. It illustrates advancements in critical aspects such as localization (box loss), classification (cls loss), and the management of complex cases (DFL loss). The training graphs indicate a consistent decline in all loss metrics, which suggests an enhancement in the model's performance as it assimilates the data. Notably, the Train Box Loss reduces from approximately 1.75 to below 0.5, while the Train Classification Loss decreases from around 1.5 to 0.25, reflecting significant progress in the model's capacity to accurately detect and classify objects. Additionally, the training precision and recall metrics show improvement, with precision nearing 0.6 and recall achieving 0.5 by the 60th epoch, which indicates a reduction in false positives and an increased rate of relevant object detection.

4.4 Testing Result

The model was tested on various datasets to evaluate its ability to perform consistently across different scenarios. A custom Python script was developed and executed in Visual Studio Code (VS Code) to deploy the model, as shown in Figure 4.10. The script processed video footage frame by frame, identifying tripping actions.



```python
import cv2
from ultralytics import YOLO
from sklearn.metrics import precision_recall_fscore_support

# Load the YOLO model
model = YOLO('models/best (11).pt')
threshold = 0.7

# Load video
cap = cv2.VideoCapture('input_videos/EPL/clip_5.mp4')
ret, frame = cap.read()
if not ret:
    print("Error: Unable to read video file")
    cap.release()
    exit()

# Define output video
out = cv2.VideoWriter('output_video.mp4', cv2.VideoWriter_fourcc(*'mp4v'), 30, (frame.shape[1], frame.shape[0]))

# Ground truth annotations (adjust for your video)
ground_truth = [0] * 30 + [1] * 11 + [0] * 59  # Example for 100 frames
true_labels, predicted_labels = [], []

frame_idx = 0
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
```

```
Frame 10: Ground Truth: 0, Predicted: 0
Frame 11: Ground Truth: 0, Predicted: 0
Frame 12: Ground Truth: 0, Predicted: 0
Frame 13: Ground Truth: 0, Predicted: 0
Frame 14: Ground Truth: 0, Predicted: 0
Frame 15: Ground Truth: 0, Predicted: 0
Frame 16: Ground Truth: 0, Predicted: 0
```

Figure 4: VS Code Deployment for Tripping Detection

The pre-trained YOLOv8 model (models/best.pt), specifically fine-tuned for tripping detection, was utilized in conjunction with Python and OpenCV to analyze MP4 format videos. The analysis was conducted on a frame-by-frame basis, employing various confidence thresholds (0.5, 0.7, 0.8, and 0.9) to eliminate low-confidence detections. Detected regions were highlighted with bounding boxes, and frames, where tripping actions were recognized in at least three out of the last five frames, were labeled as "Tripping Detected." This approach enhanced detection accuracy while reducing the occurrence of false positives.

The resulting frames were compiled into a new video file (output_video.mp4) to visually showcase the detection outcomes. Additionally, the script incorporated error-handling mechanisms to ensure seamless execution, even in cases where the input video could not be accessed. Evaluating the model at various confidence thresholds offered valuable insights into achieving a balance between sensitivity (capturing a greater number of tripping actions) and specificity (minimizing false positives). This implementation highlighted the model's effectiveness in identifying tripping actions within real-world contexts

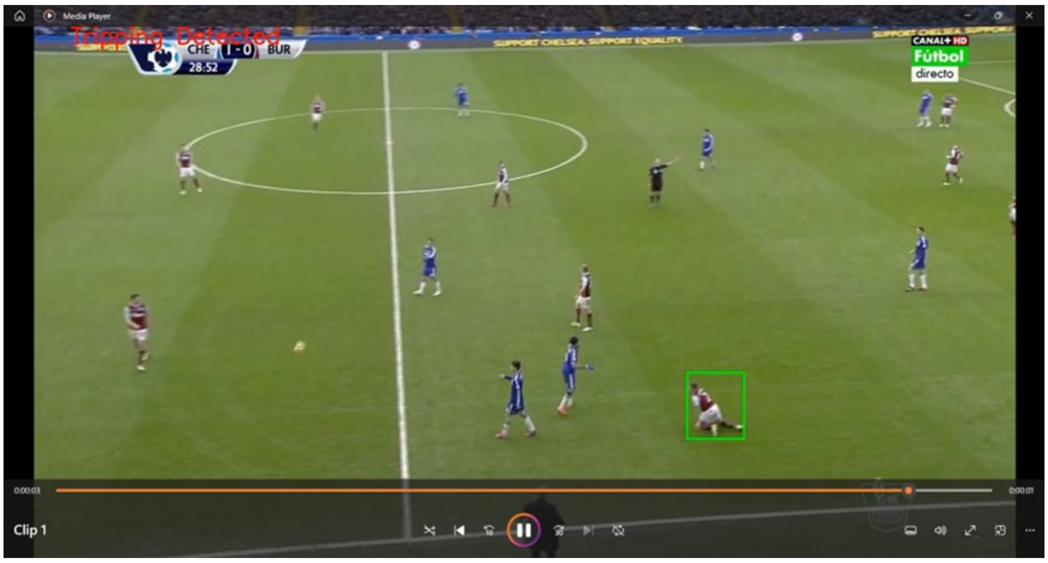


Figure 5: VS Code Deployment for Tripping Detection

Figure 5 presents the outcomes of the deployment process. In the resulting video, players engaged in tripping actions are marked with green bounding boxes, clearly indicating the identified individuals. The phrase "Tripping Detected" appears in the upper-left corner of the frame, offering explicit feedback upon the recognition of a tripping event.

The YOLOv8 model analyzed each frame of the video, detecting potential tripping actions through bounding box predictions that met or exceeded the established confidence score threshold. As the video progressed, the bounding boxes and corresponding labels were updated in real time to reflect the ongoing detections. This output underscores the model's practical efficacy in recognizing and highlighting tripping actions in real-time situations.

4.5 Testing on EPL and Non EPL Footage

The YOLOv8-based system was rigorously tested on both EPL and non-EPL footage to evaluate its robustness and adaptability across diverse conditions. Testing involved processing short-duration video clips, where the system analyzed each frame to detect tripping fouls. The system consistently identified tripping events in high-quality EPL footage, leveraging its annotations to achieve clear detection results. However, in non-EPL videos, which often presented challenges like lower resolution and inconsistent lighting, detection rates were slightly lower, especially at higher confidence thresholds.

Table 2 : Hardware and software

Confidence Threshold	EPL Detection Rate	Non-EPL Detection Rate
0.5	High	Moderate
0.7	Moderate	Low
0.8	Moderate	Low
0.9	Low	Very Low

Detection performance varied across confidence thresholds, with higher thresholds reducing false positives but occasionally missing less distinct tripping actions. The balance between sensitivity and specificity suggests the need to optimize the threshold based on the desired application.

Table 2: Hardware and software

Metric	Value	Metric
Precision	71.1%	Precision
Recall	81.8%	Recall
F1-Score	76.1%	F1-Score
mAP@50	~70%	mAP@50

These results underscore the reliability of YOLOv8 in automating foul detection and highlight its capability to generalize across varying match scenarios. Higher confidence thresholds reduced false positives but sometimes missed subtle tripping actions, demonstrating a trade-off between sensitivity and specificity [9].

5. Conclusion

This study demonstrates the feasibility of using deep learning to automate foul detection in football. By leveraging YOLOv8, the system reduced human error, promoted fair play, and provided actionable insights for post-match analysis. While the system achieved significant milestones, future improvements could focus on multi-angle detection, real-time applications, and expanding the scope to include other foul types. These advancements would further enhance its utility in sports analytics and solidify its role in modern football.

Acknowledgement

I would like to thank my supervisor and everyone who supported me throughout this project. Their guidance and encouragement were invaluable in achieving the objectives of this study.

References

[1] B. E. S., “The most popular sports in the world,” World Atlas. Accessed: Nov. 13, 2024. [Online]. Available: https://www.worldatlas.com/articles/what-are-the-most-popular-sports-in-the-world.html
[2] M. Skauge, “Five reasons why VAR is doomed to failure,” idrottsforum.org. Accessed: Apr. 13, 2024. [Online]. Available: https://idrottsforum.org/feature-skauge191004/
[3] A. Işın and Q. Yi, “Does video assistant referee technology change the magnitude and direction of home advantages and referee bias? A proof-of-concept study,” BMC Sports Sci Med Rehabil, vol. 16, no. 1, Dec. 2024, doi: 10.1186/s13102-024-00813-9.
[4] Roboflow: Build AI with Data, “Roboflow.” Accessed: Jun. 02, 2024. [Online]. Available: https://roboflow.com.
[5] Roboflow, “RoboflowImage Augmentation,” Roboflow Documentation. Accessed: Jun. 21, 2024. [Online]. Available: https://docs.roboflow.com/datasets/image-augmentation.
[6] Google Colaboratory, “Google Collab.” Accessed: Feb. 02, 2024. [Online]. Available: https://colab.research.google.com
[7] Visual Studio Code, “visual studio code.” Accessed: Jun. 02, 2024. [Online]. Available: https://code.visualstudio.com
[8] OpenCV, “OpenCV Open Source Computer Vision Library.” Accessed: Jun. 02, 2024. [Online]. Available: https://opencv.org.
[9] M. Guimarães et al., “Predicting Model Training Time to Optimize Distributed Machine Learning Applications,” Electronics (Switzerland), vol. 12, no. 4, Feb. 2023, doi: 10.3390/electronics12040871.