Assistant for the Detection of Potential Cheating Behavior in Synchronous Online Programming Exams

Francisco Ortin, Alonso Gago, Jose Quiroga, and Miguel Garcia
Computer Science Department, University of Oviedo, Oviedo, Spain
{ortin, UO269424, quirogajose, garciarmiguel}@uniovi.es

ABSTRACT

Online learning has enhanced accessibility in education, but also poses significant challenges in maintaining academic integrity during online exams, particularly when students are prohibited from accessing unauthorized resources through the Internet. Nonetheless, students must remain connected to the Internet in order to take the online exam. This paper presents a machine-learning-based assistant designed to assist instructors in detecting the use of unauthorized resources that may involve cheating during online programming exams. The system employs a convolutional neural network, followed by a recurrent neural network and a dense layer, to analyze sequences of screenshot frames from students' screens. The model achieved 95.18% accuracy and an F_2-score of 94.2%, with a focus on recall to prioritize the detection of cheating while minimizing false positives. Notably, data augmentation and class-weight adjustments significantly enhanced the model's performance, whereas transfer learning and alternative loss functions did not yield additional improvements. Although human oversight is still necessary to verify and act upon flagged activities, the system demonstrates the potential of machine learning to support real-time monitoring in large-scale online exams.

Keywords

Machine learning, cheating detection, online exams, convolutional neural network, synchronous programming exams

1. INTRODUCTION

The rapid growth of online learning has transformed traditional educational models, offering a more flexible and inclusive approach to education [1]. Distance education powered by digital tools provides significant advantages, such as removing geographic barriers, allowing students to participate in courses globally, and granting learners from diverse backgrounds access to high-quality education without needing to be physically present on a campus.

Synchronous online lecturing typically relies on web conferencing platforms like Microsoft Teams, Zoom, Google Meet, or BigBlueButton, which enable real-time communication, screen sharing, and interactive discussions [2]. However, delivering effective distance laboratory sessions (particularly synchronous programming labs) can be challenging, as strong lecturer-student interaction is necessary [3]. In these settings, instructors need to monitor code written by students in real time, addressing mistakes and deviations promptly, much like in traditional face-to-face labs.

Francisco Ortin, Alonso Gago, Jose Quiroga, and Miguel Garcia. Assistant for the detection of potential cheating behavior in synchronous online programming exams. In Caitlin Mills, Giora Alexandron, Davide Taibi, Giosuè Lo Bosco, and Luc Paquette (eds.) Proceedings of the 18th International Conference on Educational Data Mining, Palermo, Italy, July, 2025, pp. 373–380. International Educational Data Mining Society (2025).

https://doi.org/10.5281/zenodo.15870282

To address these challenges, the authors developed an infrastructure for delivering remote synchronous programming labs over the Internet in response to the COVID-19 pandemic [3]. The system allows students to attend online classes from their own computers while the instructor monitors students' activities. This setup provides instant feedback, improving student engagement and satisfaction, and has been successfully applied to various distance flipped learning environments [4].

Our infrastructure replicates the face-to-face programming lab experience in a synchronous online format. It includes a computer monitoring system, a virtual private network (VPN) for remote access, and scripts to simplify setup and system management for students. The system integrates a web conferencing platform, along with additional scripts to support instructors in managing programming labs. Our remote synchronous infrastructure has already been published in [3].

A significant challenge in distance learning is maintaining academic integrity during online exams [5]. In some cases, students are prohibited from connecting to the Internet during exams to prevent access to unauthorized resources or cheating tools. However, this creates a paradox, as students need to be online to take exams but should be restricted from accessing certain materials. Various online proctoring systems have been developed to monitor student behavior in order to detect exam integrity violations [6]. More advanced systems use machine learning (ML) algorithms to analyze student behavior patterns and flag potential instances of cheating for review by human proctors [7]. However, these systems typically rely on real-time video surveillance and do not analyze the students' activities within their computers.

In addition to webcam activation and real-time supervision, our infrastructure supports continuous monitoring of students' work throughout online exams. However, for large courses with enrollments up to of 150 students, it becomes increasingly difficult for instructors to detect fraudulent activities during exams.

To tackle this challenge, we developed a machine learning-based system to assist instructors in identifying potential cheating instances during synchronous online programming exams. This ML assistant analyzes screenshot frames from students' screens, captured by our infrastructure, and alerts the instructor to potentially fraudulent activity. If suspicious behavior is detected, the instructor is shown the screenshot sequence (Figure 1). They can either dismiss the alert if it is a false positive or take action, such as sending a popup message to the student via the remote infrastructure.

The primary contribution of this work is the development and implementation of the mentioned ML-based assistant for potential cheating detection that helps instructors monitor large-scale online exams, safeguarding academic integrity and reducing the risk of cheating. The system requires only a computer with an Internet connection, though it can be integrated with other proctoring systems

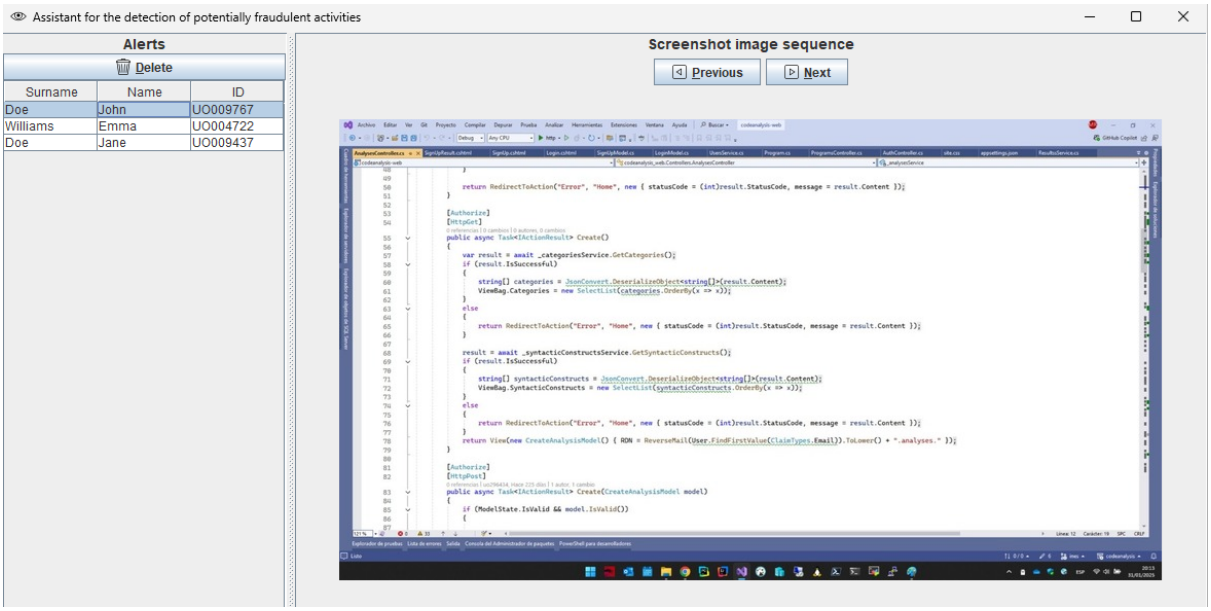

Figure 1. Machine learning assistant showing alerts of potentially cheating actions.

that use additional hardware to monitor students' actions and environments during exams.

2. RELATED WORK

Migut et al. conducted preliminary experiments where screen videos of four exams were recorded from two students, with volunteers intentionally introducing unauthorized actions according to a protocol [8]. Their system detects changes in screen content, such as application switches, by comparing visual similarity between successive frames. This approach reduces the need for full video review. Then, the next steps will involve annotating fraudulent frames and training machine learning (ML) algorithms to further reduce suspicious frames. However, the system remains a proposal without implementation or results [8].

Smirani and Boulahia proposed a system based on a convolutional neural network (CNN) that, similar to our work, does not require a webcam [9]. The system uses personal and geographical information along with screenshots from the Blackboard Learning Management System (LMS) to detect cheating, achieving 98% accuracy. However, their system only detects Internet browsing, with no real-time intervention. Consequently, it is unable to detect the use of instant messaging applications such as Discord or WhatsApp, which students may use to communicate during assessments. Additionally, the system does not support direct communication with students regarding potential violations. Finally, students are restricted to completing tests exclusively within the LMS environment.

Many online proctoring systems monitor student actions and environments during exams to detect fraud [6]. Unlike our system, these approaches use technologies such as identity authentication, webcam monitoring, and keystroke recognition to track behavior and detect suspicious activity. One common approach is camera monitoring. Luan et al. [10] proposed a system that uses two cameras—one to capture the student's face and another to record their body and surroundings. By using a trained pose recognition model, the system can efficiently classify student actions as suspicious or not. ProctorExam enhances spatial controls with 360-degree monitoring, including webcam, screen-sharing, and smartphone cameras to observe everything around the student [11]. Atoum et al. use a combination of one webcam, one wearable camera, and a microphone to monitor both the visual and acoustic environment of the test location [5].

Machine learning techniques have been applied to detect cheating. Gopane et al. trained a deep learning model to detect patterns in students' head and eye movements, achieving an F_1-score of 0.94 [7]. Their system combines a FaceNet-based face authentication module with Lucas-Kanade optical flow tracking and Active Appearance Models (AAM) for gaze estimation. This hybrid architecture enables the system to capture micro-expressions, eye blinks, gaze direction, and abnormal head movements during webcam-based online assessments. The inclusion of motion detection ensures robustness against static image spoofing, and the final output is a proctoring report generated automatically, documenting infractions based on predefined behavioral indicators. This approach highlights the efficacy of AI-driven, real-time proctoring systems for maintaining academic integrity during remote exams.

Examus collects behavioral data from online lectures to improve proctoring during exams [12]. The platform integrates with learning management systems and uses AI-driven analysis of webcam feeds, screen activity, and user behavior to detect potential academic dishonesty. By continuously monitoring facial expressions, gaze direction, and environmental cues, Examus provides real-time alerts and detailed post-exam analytics to instructors.

ProctorNet uses a pre-trained Inception CNN model to detect suspicious behaviors based on eye gaze and mouth movements [13]. Building upon this, the system integrates Inception-ResNet v1 blocks to enhance face recognition accuracy and utilizes Attentive Net for aligned face detection in video frames. These modules operate in tandem on all recognized faces, providing continuous monitoring for actions such as looking away from the screen or engaging in conversation. Alerts are generated whenever the system detects behavior that deviates from expected norms.

Some proctoring systems also incorporate student authentication methods. For example, ProctorU requires students to present ID cards to the webcam for authentication and maintain an uninterrupted audio-visual connection with the proctor throughout the exam [14]. Joshy et al. implemented a three-factor authentication scheme based on face recognition, one-time password verification, and fingerprint authentication [15].

TeSLA focuses on biometric verification for online tests, including facial and voice recognition, as well as keystroke and fingerprint analysis, to prevent impersonation and ensure the test-taker is the one providing the answers [16]. Other biometric technologies, such as fingerprint scanning, iris, retina and hand scanning, and facial recognition, are also used in online proctoring systems [17]. These approaches differ significantly from ours, as our system requires only a computer with an Internet connection and the straightforward installation of our infrastructure to support remote synchronous programming labs.

3. ARCHITECTURE OF THE ASSISTANT

Figure 2 illustrates the architecture of the assistant designed to detect potentially fraudulent activities during online exams. Multiple students may participate in an exam, connected to our remote synchronous infrastructure via the Internet. The infrastructure captures one 50x50-pixel screenshot per second of each student's activity, storing a sequence of three consecutive frames as input for the machine learning assistant. The assistant analyzes these input sequences (rather than individual screenshots) to detect potentially fraudulent behavior, generating an alert (Figure 1) only when suspicious activity is first detected after a previously correct screenshot. This approach helps minimize unnecessary alerts, making it easier for the instructor to monitor the students effectively.

Once a sample of three consecutive frames is collected for a given student, the assistant passes the sample to the ML model, which has been trained to identify potentially fraudulent activities (as described in Section 4). This model is deployed as a web API implemented in Flask, with the model loaded into memory at startup to optimize inference performance. The model functions as a binary classifier, returning "1" if fraudulent activity is detected and "0" otherwise.

If potentially fraudulent activity is detected, the assistant triggers an alert, as shown in Figure 1, accompanied by a beep sound. The instructor should then revise the screenshots that triggered the alert and decide on the appropriate action. In the case of a false positive, the alert can be dismissed. Otherwise, the student is warned through a popup message sent by the instructor via the remote synchronous infrastructure or through an oral warning. In cases of repeated fraudulent behavior, further actions may be taken, such as suspending the student's ability to continue the exam or escalating the issue to academic authorities for further investigation. Additionally, the suspicious activity frames recorded by the infrastructure are saved in their original image size, providing a detailed record for both the student and the institution to reference, if necessary.

4. METHODOLOGY

4.1 Context

The system was utilized for online exams in the "Programming Technology and Paradigms" course of the Software Engineering degree at the University of Oviedo, Spain [18]. In this course, students learn object-oriented and functional programming paradigms, concurrent and parallel programming, and basic meta-programming concepts in dynamic languages [19]. The course consists of

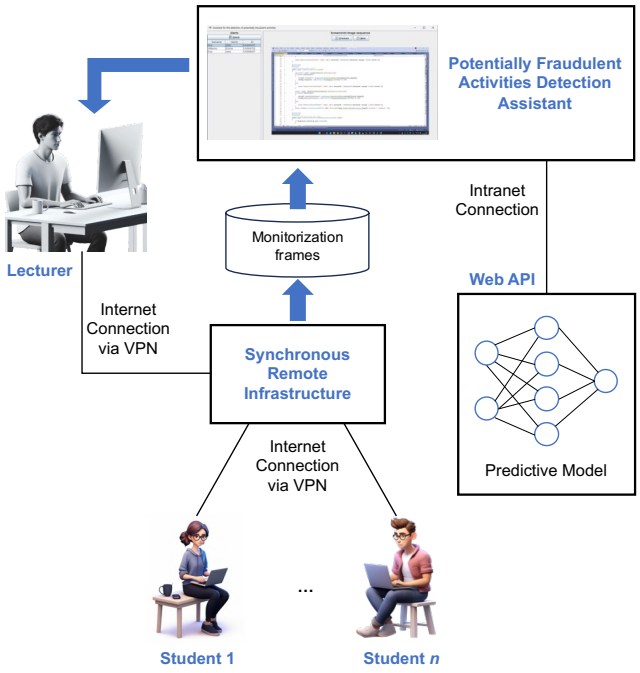

Figure 2. Architecture of the assistant designed to detect potentially cheating during online exams.

58 class hours (30 hours for programming labs and 28 hours for lectures) during the second semester, earning 6 ECTS credits. The exams involve programming tasks in C#, incorporating both object-oriented and functional paradigms, with an emphasis on sequential and concurrent programming approaches.

4.2 Dataset

To create the dataset, we recorded the activity of various students. Most of the time, students performed permitted activities with a high variance in screenshots, such as writing code, browsing the desktop, navigating files, and editing text. The system was designed to account for variations such as different desktop configurations, light/dark mode settings, system colors, and screen resolutions. By accounting for these variations, the system can better differentiate between legitimate user actions and potential fraudulent behavior, minimizing false positives and improving overall accuracy.

To enrich the dataset, we simulated common fraudulent activities prohibited during exams, such as Internet browsing, use of generative AI tools, unauthorized device interactions, and seeking external assistance via messaging or video calls. These simulated behaviors were carefully crafted to resemble real-world cheating scenarios, enabling us to train and evaluate the model's ability to detect a variety of violations in a controlled environment.

After collecting the screenshot frames, we need to label the image sequences. For that, we implemented a simple but useful sequence labeling Java application. This application facilitated the rapid labeling process by allowing users to navigate through the image sequences and categorize them as either cheating or non-cheating behavior. The labeled dataset was then used for training the model.

The final dataset comprised 8,329 sequences, each consisting of three 50x50 pixel images with three color channels (5,830 sequences for training and 2,499 for testing). To improve model performance, we augmented the training dataset using six different transformations: flipping, rotation, zooming, cropping, and modifications to contrast and brightness. This resulted in an expanded training dataset of 40,810 sequences. Since the training dataset was imbalanced, we applied undersampling to the majority class (non-cheating), achieving a balanced dataset of 36,702 samples.

4.3 ANN Models

Figure 3 illustrates the architecture of the artificial neural network (ANN) used to build the model. Each frame in the input sequence is passed through a series of convolutional neural network (CNN) layers. The number of layers, as well as the number of convolutional filters, kernel sizes, and strides, are hyperparameters defined in Section 4.5. The number of convolutional filters doubles with each subsequent layer, following conventional design principles [20].

After processing the frames through the CNN layers, the sequence is fed into a recurrent neural network (RNN) to capture the temporal dependencies between frames. Specifically, we employed either a Long Short-Term Memory (LSTM) or a Gated Recurrent Unit (GRU) network (another hyperparameter) to see which better captured the dependencies in the input sequence. The number of recurrent units in the RNN is another hyperparameter.

To prevent overfitting, a dropout layer is applied following the RNN layer, randomly setting a fraction of the recurrent units to zero during training. The dropout rate is a hyperparameter fine-tuned to balance model complexity and generalization performance. The output is then passed through a fully connected (dense) layer that aggregates the features learned by the CNN and RNN components. Finally, a sigmoid activation function is used to produce a binary classification output (cheating or non-cheating).

We also employed transfer learning to potentially enhance model performance by leveraging pre-trained weights from an image classification task [21]. Specifically, we tested the performance of replacing the convolutional layers in Figure 3 with the pre-trained models detailed in Section 4.5. The final classification layer from these pre-trained networks was excluded, and the weights were frozen to retain low-level feature representations. The remaining layers (RNN, dropout, and dense) were trained from scratch. This approach helps the model benefit from previously learned low-level features, reducing the need for a large training dataset.

4.4 Evaluation Metrics

The training set was used for model training, while the validation set was used for hyperparameter tuning. Model evaluation was performed on the test set, which was kept separate from both the training and validation sets.

We used the classical accuracy, precision and recall measures for our binary classification problem. We also used F_1- and F_2-scores, both based on F_β-score, the weighted harmonic mean of precision and recall depicted in Equation 1. F_1-score gives equal weight to both precision and recall (β=1), while F_2-score gives twice the weight on recall (β=2), reflecting the higher cost of false negatives (missing fraudulent activities). F_2-score is particularly relevant in this model, where detecting fraudulent activities is more critical than minimizing false positives.

$$F_\beta - \text{score} = (1+\beta)^2 \frac{precision \cdot recall}{\beta^2 \cdot precision + recall} \quad (1)$$

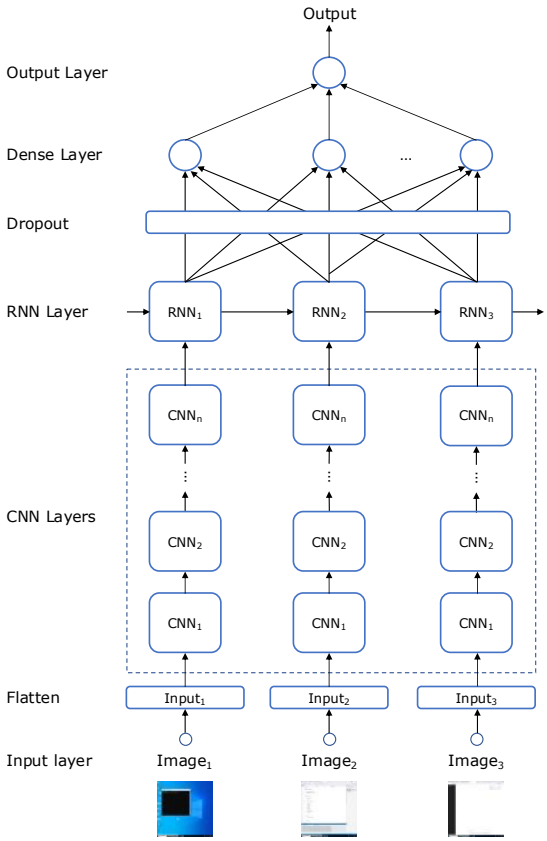

Figure 3. ANN with CNN, RNN, dropout, and dense layers.

4.5 Hyperparameter Search and Training

The two topologies identified involve several hyperparameters that significantly influence the network's learning and generalization ability. Some hyperparameters pertain to the architecture (e.g., the number of layers, neurons per layer), while others relate to the training process (e.g., batch size, learning rate).

To leverage transfer learning, we experimented with several pre-trained CNN models known for their good performance with small images: ResNet50 [22], MobileNetV2 [23], EfficientNetB0 [24], NASNetMobile [25], VGG16, and VGG19 [26]. These models were pre-trained on the ImageNet dataset, allowing us to use their learned low-level feature extraction capabilities. The selection of these models was based on their established success in a variety of computer vision tasks, particularly image classification, and their diverse architectural approaches, ranging from deep residual networks to lightweight models designed for mobile devices.

We conducted an exhaustive grid search to identify the best hyperparameter combinations (Table 1). In each iteration, the model was trained with the training set using a specific hyperparameter set, and performance was assessed using the F_2-score on the validation set (Equation 1). We explored various optimizers (Adam, RMSprop, Adamax) to determine the best one for training.

Two training approaches were tested to improve the F_2-score:

1. Assigning twice the weight to the positive class (cheating) to encourage the model to minimize false negatives [27]. This adjustment encourages the model to be more sensitive

to detecting fraudulent activities, even at the cost of increasing false positives (Section 4.4).

2. Using an alternative loss function. In addition to the classical binary cross-entropy loss function for binary classification problems, we used a differentiable surrogate loss function designed to optimize F_2-score [28].

After completing the hyperparameter search process, we selected the set of hyperparameters that yielded the best performance. Table 1 provide the values used for each hyperparameter in both models (with and without transfer learning), along with the values corresponding to the best-performing configurations (bold font). For each iteration of the search, the models were trained for a maximum of 10 epochs, with early stopping applied based on F_2-score performance and patience of two epochs.

Regarding initialization methods, we applied Xavier (Glorot) initialization to the convolutional layers and the final output dense layer [29], and He initialization to the recurrent and hidden dense layers [30].

Table 1. Best hyperparameters found for the two different artificial neural networks used.

	Hyperparameter	Values
CNN + RNN	Convolutional layers	**1**, 2, 3
	Convolutional filters	4, **8**, 16, 32
	Convolutional kernel size	**3**, 4, 5
	Strides	**1**, 2
	Convolutional activation function	**ReLU**, Leaky ReLU
	RNN cells	**LSTM**, GRU
	RNN activation function	**Tanh**, ReLU
	RNN units	16, 32, **64**, 128
	Hidden dense layer	0, **1**, 2
	Units in hidden dense layer	16, 32, **64**, 128
	Hidden layer activation function	**ReLU**, Leaky ReLU
	Dropout rate	0.0, 0.2, **0.5**
	Learning rate	10^{-4}, $\mathbf{10^{-3}}$, 10^{-2}, 10^{-1}
	Batch size	32, **64**, 128
	Optimizer	Adam, **RMSprop**, Adamax
Transfer Learning	Pre-trained model	**EfficientNetB0**, MobileNetV2, NASNetMobile, ResNet50, VGG16, VGG19
	RNN cells	**LSTM**, GRU
	RNN activation function	**Tanh**, ReLU
	RNN units	16, **32**, 64, 128
	Hidden dense layer	**0**, 1, 2
	Units in hidden dense layer	16, **32**, 64, 128
	Hidden layer activation function	**ReLU**, Leaky ReLU
	Dropout rate	0.0, **0.2**, 0.5
	Learning rate	10^{-4}, $\mathbf{10^{-3}}$, 10^{-2}, 10^{-1}
	Batch size	**32**, 64, 128
	Optimizer	**Adam**, RMSprop, Adamax

Bold font denotes the hyperparameter value selected during the hyperparameter search.

Once the optimal set of hyperparameters was determined, we conducted a final fine-tuning training session. In the case of transfer learning, the weights of the pre-trained models were unfrozen to enable fine-tuning on the target task. This adjustment ensured that the pre-trained features were adapted to better align with the specific domain of the problem. The stopping criterion for fine-tuning in both topologies was based on an increase in validation loss over three consecutive epochs. Additionally, the learning rate was dynamically reduced by a factor of 0.2 if the validation loss did not improve in the last epoch. The model exhibiting the best F_2-score performance on the validation set was selected for final evaluation with the test set.

To assess the statistical significance of the results, we employed bootstrapping with 10,000 repetitions to calculate 95% confidence intervals for each metric [31]. This process involved repeatedly sampling with replacement from the test set to generate multiple resampled datasets. The confidence intervals for the average of each metric were then computed across all resampled datasets, enabling us to evaluate whether there were statistically significant differences between the compared systems [32].

5. EVALUATION

5.1 Results

Table 2 presents the performance of the best models. The ANN architecture shown in Figure 3, which combines CNN, RNN, and dense layers, achieves the best results based on the hyperparameters listed in Table 1. For accuracy, F_1-score, and F_2-score, this model shows no statistically significant difference when compared to the EfficientNetB0 transfer-learning approach, as indicated by the overlap in their 95% confidence intervals. However, recall is significantly higher for the CNN + RNN model. Precision is the only metric in which EfficientNetB0 outperforms CNN + RNN. A notable distinction between the two models is the number of parameters: EfficientNetB0 requires 3.4 million more parameters—2.6 times the number needed by CNN + RNN.

5.2 Discussion

As shown in Table 2, transfer learning did not yield a significant performance improvement compared to our CNN + RNN neural architecture (Figure 3). Additionally, Table 2 illustrates that the models with a larger number of parameters, except for EffcientNetB0, tend to perform worse. This may be attributed to the small image size (50x50 pixels) used in our dataset, which likely causes the pre-trained models to overfit due to the limited data available. The choice of small image dimensions was necessary to ensure the assistant could process all images generated by the remote synchronous infrastructure in real time. Larger images would have hindered real-time processing. Furthermore, the CNN + RNN model, with the fewest parameters (1.3 million), facilitates faster inference, contributing to the system's ability to process frames in real time.

Our evaluation focused on the F_2-score metric, which gives twice the weight to recall compared to precision. By prioritizing recall, the F_2-score emphasizes detecting fraudulent activities while also accounting for precision. The goal is to minimize undetected cheating behaviors, even if it means the instructor must review and dismiss a few false alarms.

Several strategies were employed to optimize the F_2-score performance of our model. Data augmentation played a significant role, leading to a 31.3% improvement in this metric. This technique not only enriched the training data but also helped balance the training set, yielding very favorable results. Additionally, increasing the weight of the positive class during training resulted in a further 4.35% gain in the F_2-score. We also explored using a differentiable F_2-score surrogate loss function [28], but it did not outperform the binary cross-entropy loss.

An important feature of the assistant is its ability to facilitate the rapid detection of cheating activities. This allows instructors to

Table 2. Models with the highest performance metrics for the methodology described in Section 4.

Model	Accuracy	Precision	Recall	F_1-score	F_2-score	Parameters (millions)
CNN + RNN	**0.9518** ± 1.7%	0.9147 ± 1.6%	**0.9710** ± 1.2%	**0.9420** ± 1.2%	**0.9592** ± 1.1%	1.3
EfficientNetB0	**0.9547** ± 0.8%	**0.9352** ± 1.4%	0.9580 ± 1.0%	**0.9465** ± 2.0%	**0.9534** ± 0.5%	4.7
MobileNetV2	0.9314 ± 1.5%	0.9010 ± 1.3%	0.9362 ± 1.7%	0.9183 ± 1.6%	0.9289 ± 0.6%	2.9
NASNetMobile	0.9007 ± 1.3%	0.8123 ± 1.8%	0.9482 ± 1.5%	0.8750 ± 0.7%	0.9175 ± 1.1%	4.8
ResNet50	0.8847 ± 1.1%	0.8055 ± 1.5%	0.9147 ± 0.7%	0.8566 ± 1.9%	0.8906 ± 0.7%	24.6
VGG16	0.8891 ± 1.8%	0.8259 ± 0.5%	0.9064 ± 1.1%	0.8643 ± 1.2%	0.8891 ± 0.5%	14.8
VGG19	0.8686 ± 1.5%	0.8123 ± 1.3%	0.8718 ± 1.8%	0.8410 ± 1.3%	0.8592 ± 1.4%	20.1

Average values of the 95% confidence intervals are shown in each cell. 95% confidence intervals values are presented as percentages in each cell with ±. Bold values indicate the highest performance for each metric (with ties in cases of overlapping confidence intervals). The final column lists the number of trainable parameters (in millions).

promptly warn students about potential consequences, thereby preventing the continuation of fraudulent behavior. Such real-time intervention is undoubtedly more effective than merely storing frames and penalizing students after the exam has concluded.

Our system serves as an assistant for detecting potentially fraudulent activities in online exams, though human intervention remains necessary. It is not a standalone cheating detection system but rather a tool to assist instructors in monitoring student activity, especially in exams with large numbers of students. The use of our remote synchronous infrastructure is required for its operation, and could even be complemented by an online proctoring system. The optimization of the F_2-score ensures that the system is highly efficient in identifying potential cheating actions. However, the responsibility of verifying whether an activity is indeed fraudulent lies with the instructor, who must then take the appropriate actions, such as issuing a warning through the remote synchronous platform.

6. CONCLUSIONS

We show how an artificial neural network, consisting of CNN layers followed by RNN and dense layers, can assist instructors in detecting potentially fraudulent activities in online exams where Internet access is restricted. The proposed system analyzes sequences of screenshot frames to identify cheating behaviors, achieving an accuracy of 95.18% and an F_2-score of 94.2%. We explored several techniques to enhance the model's performance, including transfer learning, data augmentation, class-weight adjustments during training, and an alternative loss function. Notably, data augmentation and increasing the weight of the positive class significantly improved the model's performance. The primary evaluation metric used was the F_2-score, which prioritizes the model's ability to correctly identify fraudulent activities (i.e., recall) over its precision. Our system provides valuable assistance in detecting potentially fraudulent activities in online exams, though human intervention remains necessary.

As previously noted, our model was trained on screenshot frames that depict fraudulent activity specific to the course for which it was designed. Future work will explore the extent to which this pretrained model can be adapted to other courses [33] by retraining only the final dense layer. Additionally, it would be valuable to investigate the development of a meta-model capable of adapting to various courses [34]. Meta-learning techniques, such as few-shot learning, could be explored to allow the system to generalize effectively across different subjects without the need for extensive retraining [35].

All the code used in our research, including the training, hyperparameter search, and fine-tuning of the models, the implementation of the assistant, the image sequence labeler, and the Flask Web API, along with the training and test datasets and the data files generated during the experiments presented in this article, are freely available for download at https://reflection.uniovi.es/download/2025/edm.

7. ACKNOWLEDGMENTS

This work has been funded by the Government of the Principality of Asturias, with support from the European Regional Development Fund (ERDF) under project IDE/2024/000751 (GRU-GIC-24- 070). Additional funding was provided by the University of Oviedo through its support for official research groups (PAPI-24-GR-REFLECTION).

8. REFERENCES

[1] Ally, M. 2004. Foundations of educational theory for online learning. *Theory and practice of online learning*, Athabasca University Press.

[2] Fita, A., Monserrat, J. F., Moltó, G., Mestre, E. M., and Rodriguez-Burruezo, A. 2016. Use of synchronous e-learning at university degrees. *Computer Applications in Engineering Education* 24, 6, 982-993. DOI=10.1002/cae.21773.

[3] Garcia, M., Quiroga, J., and Ortin, F. 2021. An Infrastructure to Deliver Synchronous Remote Programming Labs. *IEEE Transactions on Learning Technologies* 14, 2, 161-172. DOI=10.1109/TLT.2021.3063298.

[4] Ortin, F., Quiroga, J., and Garcia, M. 2023. A Monitoring Infrastructure to Improve Flipped Learning in Technological Courses. In *7th International Conference on Education and Distance Learning* (ICEDL, Paris, France), 1541-1548.

[5] Atoum, Y., Chen, L., Liu, A. X., Hsu, S. D. H., and Liu, X. 2017. Automated Online Exam Proctoring. *IEEE Transactions on Multimedia* 19, 7, 1609-1624 (July 2017). DOI=10.1109/TMM.2017.2656064.

[6] Noorbehbahani, F., Mohammadi, A., and Aminazadeh, M. 2022. A systematic review of research on cheating in online

exams from 2010 to 2021. *Education and Information Technologies* 27, 8413–8460. DOI=10.1007/s10639-022-10927-7.

[7] Gopane, S., Kotecha, R., Obhan, J., and Pandey R. K. 2024. Cheat detection in online examinations using artificial intelligence. *UTM Asean Engineering Journal* 14, 1. DOI= 10.11113/aej.v14.20188.

[8] Migut, G., Koelma, D., Snoek, C. G. M., and Brouwer, N. 2018. Cheat me not: automated proctoring of digital exams on bring-your-own-device. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (ITiCSE 2018).

[9] Smirani, L. K. and Boulahia, J.A. 2022. An Algorithm based on Convolutional Neural Networks to Manage Online Exams via Learning Management System Without using a Webcam. *International Journal of Advanced Computer Science and Applications* 13, 3, 290-299. DOI= 10.14569/IJACSA.2022.0130336.

[10] Luan, N.K., Ha, P.T.T., and Hung, P.D. 2022. An Automated Proctor Assistant in Online Exams Using Computer Vision. In *Visualization, and Engineering* (CDVE). DOI= 10.1007/978-3-031-16538-2_12.

[11] Turnitin. ProctorExam. https://proctorexam.com, 2025.

[12] Examus. Examus proctoring service. https://appsource.microsoft.com/en-us/product/web-apps/examus.examusproctoring, 2025.

[13] Tejaswi, P., Venkatramaphanikumar, S., and Kishore, K. V. K. 2023. Proctor net: An AI framework for suspicious activity detection in online proctored examinations. *Measurement* 206, 112266. DOI=10.1016/j.measurement.2022.112266.

[14] Milone, A. S., Cortese, A. M., Balestrieri, R. L., and Pittenger, A. L. 2017. The impact of proctoredonline exams on the educational experience. *Currents in Pharmacy Teaching and Learning* 9, 1,108–114. DOI=10.1016/j.cptl.2016.08.037.

[15] Joshy, N., Ganesh Kumar, M., Mukhilan, P., Manoj Prasad, V., and Ramasamy, T. 2018. Multi-factor authentication scheme for online examination. *International Journal of Pure and Applied Mathematics* 119, 15, 1705-1712.

[16] Draaijer, S., Jefferies, A., and Somers, G. 2018. Online proctoring for remote examination: A state of playing higher education in the EU. *Communications in Computer and Information Science* 829, 96–108. DOI=10.1007/978-3-319-97807-9_8.

[17] Mahadi, N. A., Mohamed, M. A., Mohamad, A. I., Makhtar, M., Kadir, M. F. A., and Mamat, M. 2018. A Survey of Machine Learning Techniques for Behavioral-Based Biometric User Authentication. *Recent Advances in Cryptography and Network Security*. DOI=10.5772/intechopen.76685.

[18] Ortin, F., Redondo, J. M., and Quiroga, J. 2017. Design and evaluation of an alternative programming paradigms course. *Telematics & Informatics* 34, 6, 813–823. DOI= 10.1016/j.tele.2016.09.014.

[19] Ortin, F., Redondo, J. M., and Quiroga, J. 2016. Design of a programming paradigms course using one single programming language. In Proceedings of World Conference on Information Systems and Technologies (Recife, Brazil, 2016), 179–188, DOI=10.1007/978-3-319-31307-8_18.

[20] Li, H., Yue, X., Wang, Z., Wang, W., Tomiyama, H., and Meng, L., 2021. A survey of Convolutional Neural Networks —From software to hardware and the applications in measurement. *Measurement: Sensors* 8, 100080. DOI= 10.1016/j.measen.2021.100080.

[21] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. 2014. How transferable are features in deep neural networks? In *Proceedings of the 27th International Conference on Neural Information Processing Systems* (NIPS 2014), 3320–3328.

[22] He, K., Zhang, X., Ren, S., and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (CVPR), 770-778.

[23] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L. 2018. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (CVPR), 4510-4520.

[24] Tan, M. and Le, Q. V. 2019. EfficientNet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning* (ICML), 6105-6114.

[25] Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (CVPR), 8697-8710.

[26] Simonyan, K. and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations* (ICLR), 1-14.

[27] Cui, Y., Jia, M., Lin, T. Y., Song, Y., and Belongie, S. 2019. Class-Balanced Loss Based on Effective Number of Samples. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition* (CVPR, Long Beach, CA, USA, 2019), 9260-9269.

[28] Lee, N., Yang, H., and Yoo, H. 2021. A surrogate loss function for optimization of F_β score in binary classification with imbalanced data. *ArXiv* 2104.01459, 1-17, DOI=10.48550/arXiv.2104.01459.

[29] Glorot, X. and Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics* (AISTATS) 9, 249-256.

[30] He, K., Zhang, X., Ren, S., and Sun, J. 2015. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proceedings of the IEEE International Conference on Computer Vision* (ICCV), 1026-1034.

[31] Davison, A. C. and Hinkley, D. V. 1997. Bootstrap methods and their application. *Cambridge University Press*.

[32] Georges, A., Buytaert, D., and Eeckhout, L. 2007. Statistically rigorous Java performance evaluation. In *Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications* (OOPSLA 2007, October 21-25, 2007, Montreal, Quebec, Canada), 57–76.

[33] Ortin, F., Facundo, G., Garcia, M. 2023. Analyzing syntactic constructs of Java programs with machine learning. *Expert Systems with Applications* 215, 119398-119414. DOI=10.1016/j.eswa.2022.119398.

[34] Rodriguez-Prieto, O., Pato, A., Ortin, F. 2025. PLangRec: Deep-learning model to predict the programming language from a single line of code. *Future Generation Computer*

Systems 166, 107640-107655. DOI=10.1016/j.future.2024.107640.

[35] Álvarez-Fidalgo, D., and Ortin, F. 2025. CLAVE: A deep learning model for source code authorship verification with contrastive learning and transformer encoders. *Information Processing & Management* 62, 3, 104005. DOI=10.1016/j.ipm.2024.104005.