IMPLEMENTATION OF FACE RECOGNITION AND LIVENESS DETECTION SYSTEM USING TENSORFLOW.JS

Muhammad Basurah[1], Windra Swastika[2], Oesman Hendra Kelana[3]

[1,2,3] Informatic Engineerin, FST, Ma Chung University Malang
[1]basurah.muhammad.mb@gmail.com, [2]windra.swastika@machung.ac.id, [3]oesman.hendra@machung.ac.id

Abstract

Facial recognition is a popular biometric security system used to authenticate individuals based on their unique facial structure. However, this system is vulnerable to spoofing attacks where the attacker can bypass the system using fake representations of the user's face such as photos, statues or videos. Liveness detection is a method used to address this issue by verifying that the user is a real person and not a representation. This journal article focuses on the life sign method of liveness detection, which utilizes facial movements to confirm the user's existence. We implement the latest technology of artificial intelligence from TensorFlow.js using face-api.js and compare it with the GLCM algorithm. However, even with the life sign detection method, there is still a chance of bypassing the system if an attacker uses a video recording. To mitigate this, we propose the addition of an object detection system to detect the hardware used to show video recordings with ml5.js. Our face recognition and expression detection system, using the pre-trained model face-api.js, achieved an accuracy of 85% and 82.5%, respectively, and the object detection system built with ml5.js has high accuracy and is very effective for liveness detection. Our results indicate that face-api.js outperformed GLCM algorithm in detecting spoofing attempts.

Keywords : life sign, face-api.js, Ml5.js, tensorflow, expression detection, object detection

1. Introduction

Facial recognition systems are now becoming very common and widely used in various latest technologies. Starting from facial recognition on smartphones and facial recognition for other security systems. The facial recognition system is included in the biometric security system which is still one family with voice recognition, fingerprint recognition, and retina/iris recognition (Ardiansiah, Widyadi Setiawan, & Linawati, 2016) The facial recognition system in the human body is very interesting where humans can still recognize someone even though he has grown or experienced changes, this can also be done by a computer that applies facial recognition with a good algorithm (Sudiana, Diponegoro, & Priambodo, 2018).

The problem that often arises in facial recognition systems is that this system is easily deceived by the use of fake faces or representations of human faces such as photos, videos, masks, and statues called spoofing attack. Spoofing is the act of cheating to trick the system that was built by exploiting the security holes of the system (Sthevanie & Ramadhani, 2014) (Pratap, Priya, & Mani, 2019). This is because the basic principle of face recognition is to quote unique face information, then encode it and compare it with the decoded results that were previously carried out (Li, Mu, Li, & Peng, 2020). Human face representations cannot be detected without liveness detection. Without the Liveness detection system in place, malicious users would use a photo of a person already registered with the system to bypass the facial recognition system and gain personal gain.

To make the facial recognition system on online presence more secure, we need a program that can detect whether the user is real or fake called Liveness detection. Liveness detection is one way to identify and confirm a person's identity which refers to the use of computer vision technology to detect the real presence of a living user, not representations such as photos or masks (Chakraborty & Das, 2014) (Liu, et al., 2019).

Implementing liveness detection into face recognition systems can greatly improve security. Unlike regular face recognition without liveness detection, which just takes data from the camera without checking if it's a real person, there's a potential weakness. The normal face recognition treats everything in front of it as a real human face, even if it's just a picture or an object.

That's where liveness detection comes in to fix this. It's a way to solve these issues. Liveness detection first checks if the thing in front of the camera is a real human or just a fake. If it finds out it's not a real human, it stops the face recognition, making it impossible to trick the system. By

combining liveness detection with face recognition, the system's security gets a lot stronger.

Several actions can be taken to run Liveness detection, one of which is Heuristic-based algorithms such as eye movements, lips, and eye blinks. The algorithm tries to detect lip movement to ensure the user isn't showing a photo of another person's face (because the photo doesn't move their lips or blink).

In previous studies, there have been studies that discussed liveness detection but used different methods. In a study entitled Face Liveness detection Using Dynamic Texture, texture checking of the captured image is used to distinguish which users are real or fake (using photos). In this study, there is a weakness where a lot of training results are needed and consist of 2 types, namely fake images (with photo print textures) and original images (not photos) for each user who wants to be registered (Komulainen, De Martino, & Komulainen, 2014). The application of the liveness detection system will not be far from artificial intelligence technology. Artificial Intelligence is a part of computer science that studies how to make machines do work as well as humans. Then the apply of machine learning where it is a branch of artificial intelligence that focuses on building and studying a system make it able to learn from the data it obtains (Fikriya, Irawan, & Soetrisno, 2017) (Bonetto & Latzko, 2020). Then adding computer vision so that the computer can retrieve visual data from the webcam, analyze images, videos, and then process the data to obtain the required information (Wibowo, 2016) (Tian & Wang, 2020).

All applications related to the artificial intelligence will be built with the help of the open source TensorFlow library. TensorFlow is an open source library for machine learning released by Google that supports several programming languages and can be used on all operating systems (Nugroho, Fenriana, & Arjianto, 2020) (Pang, Nijkamp, & Wu, 2020). TensorFlow was built with the aim of doing machine learning and neural networks in research. TensorFlow works by combining computational algebra and compilation optimization techniques, which facilitates the computation of many mathematical expressions. Another advantage provided by TensorFlow is that the results of the written program code will be easier to read and understand by others because it relies on libraries to run Machine Learning (Anggraini, 2020) (Abu, Indra, & Rahman, 2019) (Abadi, Barham, Chen, & Chen, 2016). in application TensorFlow through the website, we can be helped by tensorflow.js which uses the JavaScript programming language. JavaScript is a language in the form of a collection of scripts whose functions run on an HTML document, throughout the history of the internet this language was the first scripting language for the web (Sahi, 2020).

In this journal, the solution that will be given to the weakness of the face recognition system is to provide a liveness detection system using the life sign method which will be built using a pre trained model from tensorflow.js with the name face api.js. With the implementation of liveness detection, a facial recognition system will be formed with a much stronger security system against attacks.

2. Research Method

2.1 Model Architecture and Features

Face-api.js is a cutting-edge javascript module that utilizes Convolutional Neural Networks (CNNs) built on top of the tensorflow.js core to provide efficient face detection, face recognition, and face expression recognition capabilities for web and mobile applications. The module's face detection feature utilizes a Single Shot Multibox Detector (SSD) based on MobileNetV1 architecture, which can compute the location of every face in an image and return their bounding boxes along with their corresponding probabilities. The model provides high accuracy in detecting face bounding boxes and is not designed to prioritize low inference time. The size of the quantized model is approximately 5.4 MB (ssd_mobilenetv1_model), and the weights are trained on the WIDERFACE dataset and provided by yeephycho on GitHub.

The face recognition capability of face-api.js is powered by a ResNet-34-like architecture, which can compute a 128-dimensional face descriptor feature vector from any given face image. This feature vector can describe a person's facial characteristics and is not limited to the set of faces used for training, allowing for face recognition of any person. Users can compare the similarity of two arbitrary faces by comparing their face descriptors using any classifier of their choice or by computing the euclidean distance. The neural network is equivalent to the FaceRecognizerNet used in face-recognition.js and the net used in the dlib face recognition example. The weights are trained by davisking, and the model achieves an impressive prediction accuracy of 99.38% on the LFW (Labeled Faces in the Wild) benchmark for face recognition. The size of the quantized model is roughly 6.2 MB (face_recognition_model).

The face expression recognition model in face-api.js is lightweight, fast, and provides reasonable accuracy. The model employs depthwise separable convolutions and densely connected blocks and has a size of approximately 310kb. It is trained on a variety of images from publicly available datasets, as well as images scraped from the web. However, it is worth noting that wearing glasses might decrease the accuracy of the prediction results. Overall, face-api.js is a state-of-the-art tool for developers looking

to implement advanced facial recognition features in their web and mobile applications.

ML5.js is a versatile machine-learning library that has brought the power of machine learning into the web browser. Based on the popular tensorflow.js, ml5.js allows developers to create machine learning models and execute them within the browser, without the need for server-side processing. One of the most powerful features of ml5.js is its object detection method, which takes image classification to the next level. It enables the identification of multiple objects within an image or video and locates them by drawing bounding boxes around the detected content.

Object detection is a crucial computer vision technique used to identify and locate objects within an image or video. By drawing bounding boxes around the identified objects, the method helps to provide context and extract valuable information from visual data. Within ml5.js, there are two pre-trained models for object detection: YOLO and COCO-SSD. COCO-SSD is the model we will focus on here.

COCO-SSD is a type of convolutional neural network (CNN) that uses successive layers of computation to find complex patterns within an image. By analyzing pixels and identifying patterns such as edges, corners, or circles,

2.2 Data Preprocessing for Model Input

Face recognition is a crucial part of many modern applications, and while pre-trained models are available, they still require customization to recognize specific individuals. Face-api.js offers the face_recognition_model, which allows users to input new training data to recognize people already registered in a hosting platform. This process involves training the model by feeding it with labeled images to let it learn the unique features of each individual's face, enabling it to recognize them in new images. In this study, 12 face images of 6 people were manually labeled, with each person having 2 photos. The images were used to train the model, and facial image samples are shown in Figure 1.

Figure 1 Sample new face to register

Once the model has access to the images, the ResNet-34 architecture extracts face descriptors, which are numerical representations of facial features that the model learns to recognize. These descriptors are then stored in a database, and face-api.js API uses the ResNet-34 model to generate them. With the trained model and stored face descriptors, the system can recognize individuals in real-time applications.

2.3 Implementation

In order to implement liveness detection using face-api.js in your web project, there are several steps that you need to follow. Firstly, you can download or include face-api.js from the GitHub repository, or use a CDN to include it in your web page. For example, you can add the following script tag in your HTML file to load face-api.js from the unpkg CDN that can be seen in figure 2.

```
<script src="https://unpkg.com/face-api.js@0.24.2/face-api.min.js"></script>
```

Figure 2 Load face-api.js

Once you have included the face-api.js library in your web project, you can load an image using an img tag or programmatically using JavaScript. Then, you can use the detectAllFaces function provided by face-api.js to detect all the faces in the loaded image. This function returns a Promise that resolves to an array of Face Detection objects, which contain information about the location and size of the detected faces that can be seen in figure 3.

```
const img = document.getElementById('my-image')
faceapi.detectAllFaces(img).then((detections) => {
  // Do something with the detected faces
})
```

Figure 3 Load an image

In order to visualize the detected faces, you can use the Canvas API to draw bounding boxes around them. For each detected face, you can use the box property of the corresponding Face Detection object to get the coordinates of the bounding box and can be seen in figure 4.

```
const canvas = document.getElementById('my-canvas')
const context = canvas.getContext('2d')
detections.forEach((detection) => {
  const box = detection.box
  context.beginPath()
  context.rect(box.x, box.y, box.width, box.height)
  context.stroke()
})
```

Figure 4 Visualize

If you want to perform face recognition, you can use the computeFaceDescriptor function provided by face-api.js to extract the face descriptor from each detected face. This function returns a Promise that resolves to a Float32Array of 128 values that represent the unique features of the face. These features can be used for face matching or liveness detection purposes that can be seen in figure 5.

```
faceapi.computeFaceDescriptor(img, detection).then((descriptor) => {
    // Do something with the face descriptor
  })
```

Figure 5 Perfome face recognition

By following these steps, you can implement face-api.js to detect faces in an image, draw bounding boxes around them, and extract face descriptors for implementing liveness detection in your website. These features can be used for various applications such as facial authentication, fraud detection, and security systems.

Face recognition is now possible by passing the extracted data and aligned face images to a face recognition network like ResNet-34. The neural network maps different facial features onto a face descriptor, which is a 128-value vector, also called face embeddings. This model is not restricted to the training dataset and can be used for face recognition of any person. The face descriptors of the extracted image are compared to the reference data using the Euclidean distance, which is the distance used to measure the similarity of pattern samples in the geometric pattern space. The Euclidean metric is widely used because it is simple to calculate and it defines whether two faces are similar based on a threshold value. The Euclidean distance for an n-dimensional space is defined as figure 6 and euclidean distance for a two-dimensional space is defined in equation 1.

$$d(p,q) = \sqrt{\sum_{i=0}^{n}(p_i - q_i)^2} = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2} \quad (1)$$

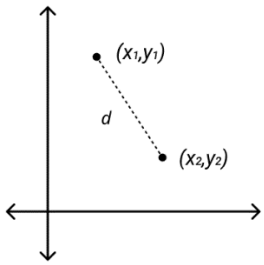


Figure 6 Euclidean distance for a two-dimensional space

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (2)$$

The traditional Euclidean distance treats each dimension of a vector equally without considering their relationship, which can limit its effectiveness and scope. To address this, weighted Euclidean distance is often used to improve the accuracy of face identification. The comparison of two face images using Euclidean distance to find similarity is demonstrated in the following pictures that can be seen in figure 8.

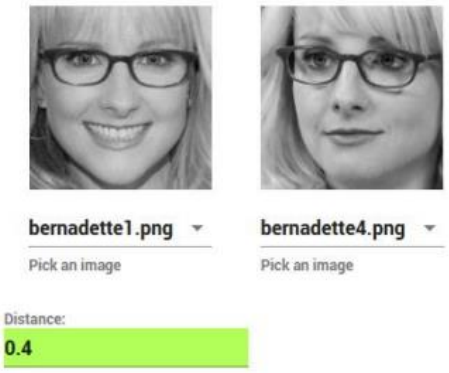


Figure 7 The comparison of two face images using Euclidean distance

After preparing all the necessary models, the following is the main flow of the results obtained from the implementation of liveness detection that can be seen in figure 9.

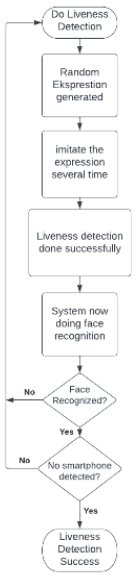


Figure 8 Overall flow for liveness detection

2.4 Hyperparameter Tuning

Hyperparameter tuning is needed to optimize the performance of a machine learning model. Hyperparameters are settings that are not learned by the model during training but instead are set by the user before training. These settings can have a significant impact on the performance of the model, and so it is important to find the optimal values for these hyperparameters in order to achieve the best possible performance. Hyperparameter tuning involves systematically adjusting the values of these settings and evaluating the resulting model performance to find the best combination of hyperparameter values for a given task.

Throughout the project, it was observed that detecting angry expressions using facial expression detection posed a challenge, resulting in an accuracy

of just over 50%. To address this issue, I decided to increase the model's sensitivity to detect angry and disgusted expressions, setting the threshold to 0.65. This led to a significant improvement in expression detection accuracy, as evidenced by the test results shown in the accompanying picture that can be seen in figure 10.

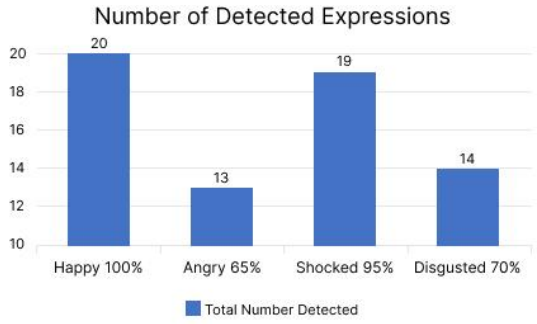
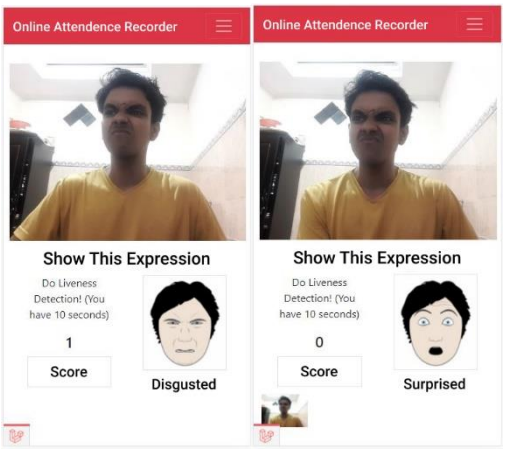


Figure 9 Hyperparameter tuning effects

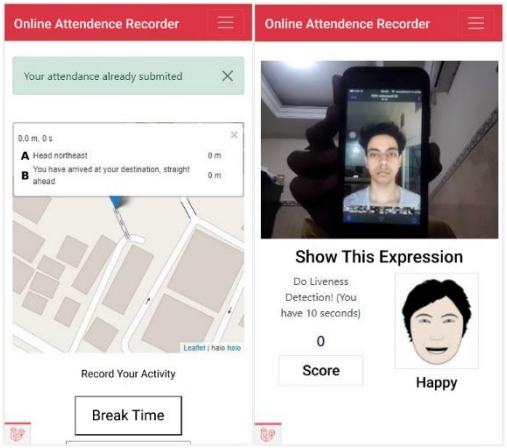
3. Results and Comparison

In this study, an arithmetic was utilized to conduct a simulation experiment. The participants were prompted to perform liveness detection five times by following random instructions generated by the system. Liveness detection was required before the participants could do face recognition to input data into the presence system. The participants were awarded a score if they followed the instructions correctly, and their score was reset to zero if they attempted to spoof the system using a photo. The results of the experiment are presented in Figure 11.

Furthermore, the accuracy of the liveness detection was assessed by conducting tests on five individuals in four different conditions for each model. The Face recognition results are displayed in Table 1, the expression detection results in Figure 12, and the object detection results in Table 2.



(a) (b)



(c) (d)

Figure 10 (a) successful liveness detection with added score; (b) getting 5 scores, resetting and redirecting to the next page; (c) inputting data to database; (d) attempting to spoof with a photo an

Table 1 Test for face recognition with 85% result accuracy

System	Recognized	Not
Face Recognition	17	3

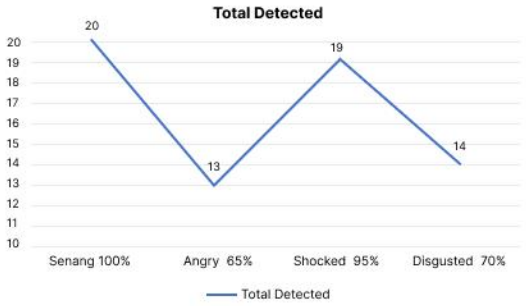


Figure 11 Expression detection accuracy result is 82.5%

Table 2 Object detection test

Condition	Object Detection	Result
1	ml5.js	Detected
2	ml5.js	Detected
3	ml5.js	Detected
4	ml5.js	Detected

The Face Recognition test results were obtained by conducting tests with 5 participants under 4 different conditions. These conditions were derived from variations in camera quality, including a 2 Mega Pixel camera for lower quality and a 12 Mega Pixel camera for higher quality, as well as lighting conditions ranging from no light (0 Lux) to high light intensity (65 Lux). The combination of camera quality and lighting conditions resulted in 4 distinct test scenarios, yielding a total of 20 test results, as shown in Table 1.

Similarly, expression detection and object detection tests were carried out across these 4 conditions, with the main difference being the number of tests performed. With the capability of our TensorFlow.js technology to recognize Happy, Angry, Shocked, and Sad expressions, and considering 5 participants in 4 distinct conditions, the expression detection tests result were performed 80 times in total. Following the comprehensive evaluation of all conditions and expression tests, observations indicated that detecting Angry expressions proved to be more challenging, particularly under low light and low camera quality conditions. This led to the successful detection of only 13 Angry expressions out of 20 attempts. Conversely, detecting Happy expressions exhibited the highest success rate, with 20 successful detections across all conditions.

For the object detection utilizing Ml5.js, an accuracy assessment was conducted to identify smartphones, a common medium for attempting to deceive face recognition systems using 2D images. The evaluation was performed under the same 4 conditions: high camera quality with high light intensity, low camera quality with high light intensity, high camera quality with low light intensity, and low camera quality with low light intensity. The results of these tests are presented in Table 2, demonstrating successful detection of smartphones in all evaluated conditions.

Comparatively, the conventional face recognition lacking liveness detection capability could only undergo the initial test, as it lacks the features to assess expressions or objects in its field of view. By incorporating liveness detection, attempts at spoofing through actions like showing a photo from a phone to simulate an expression detection can be thwarted using the object detection mechanism. Enhancing the object detection capabilities holds the potential to further reinforce liveness detection, for instance, by training the system to identify objects beyond smartphones, such as printed photos or stone textures used in face replicas, thus bolstering overall security.

For the comparison, we will compare our face-api.js web based with the GLCM algorithm non-website based that can be used to do spoofing detection on facial biometrics. GLCM algorithm that compares the texture features of a face image. GLCM generates a co-occurrence matrix of pixel intensity values and their spatial relationship in an image, which is then analyzed to extract texture features such as contrast, homogeneity, and energy. By comparing the texture features of the input face image with those of the reference image, the GLCM algorithm can detect if the input image is a spoof or a genuine face image, you can find more detail at Spoofing detection on facial images recognition using LBP and GLCM combination (Sthevanie & Ramadhani, 2014). The result of the GLCM Algorithm from that research can be seen in figure 13.

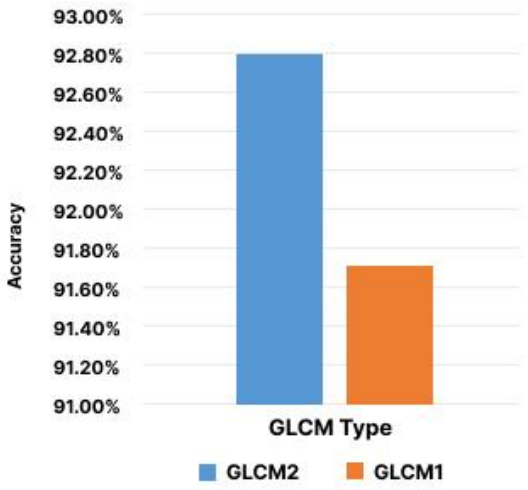


Figure 12 GLCM accuracy result

Based on Figure 13, GLCM achieved the highest accuracy at 92% for overall performance spoofing detection compared to the challenging face-api.js liveness detection, which involves face recognition, expression detection, and object detection. Further comparison will be discussed below.

Pros of using face-api.js to do liveness detection:

1. Easy to use and implement, with a clear API and good documentation
2. Offers a wide range of face detection and recognition features, including the ability to detect facial landmarks, expressions, and age and gender estimation
3. Works well in real-time applications, with fast processing speeds
4. Can be used on a variety of platforms, including web and mobile
5. can block spoofing from deep fakes or 3D-printed masks

Cons of using face-api.js to do liveness detection:

1. May not be as accurate as more complex algorithms, particularly for complex scenarios or in situations with poor lighting or low image quality
2. May require a significant amount of training data to achieve high accuracy

Pros of using GLCM algorithm to do liveness detection:

1. Has been shown to be effective for detecting spoofing attacks in some scenarios
2. Can be implemented using relatively simple image-processing techniques

3. May be particularly effective for detecting certain types of spoofing attacks, such as those involving printed images

Cons of using GLCM algorithm to do liveness detection:

1. May not be as effective for detecting more sophisticated spoofing attacks, such as those using deep fakes or 3D-printed masks
2. May require more processing time and computational resources than some other algorithms, particularly for large images or datasets
3. Limited variety of platforms

4. Conclusion

Based on the results of implementing liveness detection using Face-api.js from tensorflow.js, it can be concluded that this system has several advantages over using GLCM algorithm. The combination of face recognition and expression detection achieves an accuracy rate of 85%, with an additional liveness detection system that has an impressive 82.5% accuracy in detecting user expressions.

Moreover, the advanced security system created to prevent fraudulent activities such as replay photo attacks is also notable. By using object detection, the system can accurately detect the presence of a real user and prevent the use of fake or manipulated images.

However, there is still room for improvement in the system. Optimizing CSS and JavaScript can significantly reduce the website's rendering blocking time, improving the website's loading speed and user experience. Additionally, storing the rendering results of each face in a database system can further enhance the face recognition system's accuracy and reduce the rendering time for clients. Overall, using Face-api.js for liveness detection and face recognition provides a robust and accurate solution for online presence systems.

References:

Abadi, M., Barham, P., Chen, J., & Chen, Z. (2016). TensorFlow: A system for large-scale machine learning. *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016*.

Abu, M. A., Indra, N. H., & Rahman, A. H. (2019). A study on image classification based on deep learning and tensorflow. *International Journal of Engineering Research and Technology*.

Anggraini, W. (2020). Deep Learning Untuk Deteksi Wajah Yang Berhijab Menggunakan Algoritma Convolutional Neural Network (CNN) Dengan Tensorflow. 16.

Ardiansiah, Widyadi Setiawan, & Linawati. (2016). Sistem Pengenalan Wajah Dengan Metode Face Features. *E-Journal SPEKTRUM Vol 6, No 3*, 22.

Bonetto, R., & Latzko, V. (2020). Machine Learning. *Computing in Communication Networks: From Theory to Practice*, 5.

Chakraborty, S., & Das, D. (2014). An Overview Of Face Liveness Detection. *International Journal on Information Theory (IJIT), Vol.3*.

Fikriya, Z. A., Irawan, M. I., & Soetrisno. (2017). Implementasi Extreme Learning Machine untuk Pengenalan Objek Citra Digital. *Jurnal Sains dan Seni ITS Vol. 6, No.1*.

Komulainen, J., De Martino, J. M., & Komulainen, M. (2014). Face Liveness Detection Using Dynamic Texture. *EURASIP Journal on Image and Video Processing*, 2.

Li, L., Mu, X., Li, S., & Peng, H. (2020). A Review of Face Recognition Technology. *IEEE Access*.

Liu, S., Song, Y., Zhang, M., Zhao, J., Yang, S., & Hou, K. (2019). An identity authentication method combining liveness detection and face recognition. *Sensors (Switzerland)*.

Nugroho, P. A., Fenriana, I., & Arjianto, R. (2020). Implementasi Deep Learning Menggunakan Convolutional Neural Network (CNN) Pada Ekspresi Wajah. *Jurnal Algor*, 17.

Pang, B., Nijkamp, E., & Wu, Y. N. (2020). Deep Learning With TensorFlow: A Review. *Journal of Educational and Behavioral Statistics*.

Pratap, K., Priya, A., & Mani, G. (2019). Technologies to overcome spoofing attack in facial recognition. *International Journal of Recent Technology and Engineering*, 3.

Sahi, A. (2020). Aplikasi Test Potensi Akademik Seleksi Saringan Masuk LP31 Berbasis Web Online Menggunakan Framework Codeigniter. *Jurnal Teknologi Informasi Dan Komunikasi*.

Sthevanie, F., & Ramadhani, K. (2014). Spoofing detection on facial images recognition using LBP and GLCM combination. *International Conference on Data and Information Science*.

Sudiana, D., Diponegoro, A. D., & Priambodo, P. S. (2018). Sistem Pengenalan Wajah (Face Recognition) Menggunakan Metode Hidden Markov Model (HMM). 2.

Tian, H., & Wang, T. (2020). Computer vision technology in agricultural automation —A review. *Information Processing in Agriculture*, 2.

Wibowo, A. P. (2016). Implementasi Teknik Computer Vision Dengan Metode Colored Markers Trajectory Secara Real Time. *Jurnal Teknik Informatika Vol. 8 No.1*.