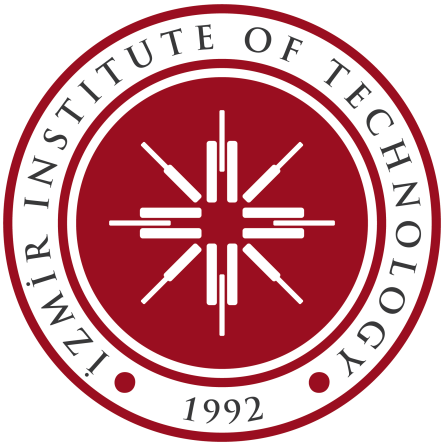
Seds 536 Image Understanding
Project Progress Report
Fall 2024

Project Title: Real-Time Gesture Recognition for Device Control

December 29, 2024



- Kübra Destebaşı & 313011012

Abstract

In this project, I created a system that uses hand gestures to control the sound settings in real time. I used two gestures: thumbs up to increase volume and thumbs down to decrease volume. I tested two models: YOLO, which is a deep learning model for detecting objects, and KNN, which is a simpler machine learning method. I used the Hagrid data set to prepare training, validation and test data. I also used Mediapipe to extract features from the images. The results show that KNN is better for accuracy, speed and using less hardware, while YOLO works better for more complex tasks. This report explains how I did the project, my experiments, and the results. It also looks at the pros and cons of each model.

1 Introduction

Controlling devices using hand gestures is becoming a widely used and practical way to interact with technology. Hand gestures offer a touch-free, user-friendly experience, making them ideal in situations where touching a device is not convenient or possible. For example, hand gestures can be helpful in scenarios where hygiene is a priority, such as in medical settings or public spaces. They are also useful for controlling devices when hands are dirty, such as in a kitchen or workshop. This project focuses on creating a system that uses hand gestures to control the volume of a device. Specifically, the system recognizes two simple gestures: thumbs up to increase the volume and thumbs down to lower it.

The goal of this project was to develop a system that works in real-time and evaluates the effectiveness of two different approaches to gesture recognition. I tested two different models: YOLO and KNN. YOLO is a very powerful deep learning model that is often used to detect objects in images. It works well for complex tasks, but it needs more hardware resources like a GPU. On the other hand, KNN is a simpler machine learning algorithm. It is easy to use, works with smaller datasets, and can run on a CPU. I decided to compare these two methods to determine their strengths and weaknesses in recognizing hand gestures effectively.

To build this system, I used the Hagrid Hukenovs (2024) dataset, which contains a large number of images of hand gestures. The dataset includes a variety of conditions, such as different lighting, hand angles, and hand sizes, distance, and different demographic features, that making it suitable for real-world applications. Focuses on two gestures: thumbs up and thumbs down. To create a smaller, more manageable dataset, I randomly sampled 14,000 images from the original dataset, with 7,000 images for each class. This smaller dataset still maintains diversity and balance, making it suitable for training and testing models. This allowed me to work with a balanced and diverse dataset while reducing computational requirements.

I prepared the dataset by dividing it into three parts: training, validation, and test sets. This way, the models could learn from a portion of the data while being tested on images they had not seen before. The raw images could not be directly used for training, so I used Mediapipe (2024a), a tool for detecting key points on the hand, such as the wrist, fingers, and fingertips.A critical part of this project was the role of Mediapipe, which acted as a pre-trained hand detection system.Mediapipe enabled the extraction of key features from the dataset, such as hand landmarks, distances, and angles. These features were crucial for both YOLO and KNN, as they provided the input data required for training and testing. For YOLO, Mediapipe helped generate bounding boxes to annotate the detected hands, while for KNN, it provided normalized landmarks and additional feature calculations.

The main objective of this project was to evaluate the performance of YOLO and KNN in the context of hand gesture recognition for real-time applications. I compared the two models based on three key factors: accuracy, speed, and hardware requirements. Accuracy determines how well the models can classify gestures correctly, speed measures the time required to make predictions, and hardware requirements highlight the type of computational resources needed to run the models efficiently. By analyzing these factors, I aimed to understand the strengths and weaknesses of each approach.

YOLO, being a deep learning model, is designed for complex tasks and is capable of detecting objects with high accuracy in images. However, it requires significant computational power, typically relying on a GPU for real-time performance. On the other hand, KNN is a simpler machine learning algorithm that is easy to implement and works well with smaller datasets. It operates efficiently on a CPU and has lower hardware requirements, making it suitable for lightweight applications.

This project compares two different approaches and shows how pre-trained systems like Mediapipe can make it easier to build gesture recognition systems. Using these tools, I learned about the differences between deep learning and traditional machine learning methods for real-world applications. These findings will help me choose the best model for similar tasks in the future, depending on the specific needs of the project.

2 Literature Review

Gesture recognition is a growing field in computer science and human-computer interaction (HCI). It allows people to communicate with machines using gestures instead of devices like keyboards or mice. The main goal is to make communication between humans and machines simple and natural Jain et al. (2022).

There are two main methods for gesture recognition: sensor-based and vision-based. Sensor-based methods use devices like data gloves with sensors to detect hand movements. These methods are very accurate but can be expensive and uncomfortable to use for long periods. They may also be hard for older adults or people with physical difficulties to use Munir Oudah et al. (2020).

Vision-based methods use cameras and computer algorithms to detect gestures. These methods are easier to use because they don't require wearing special devices. Vision-based systems use features like the shape of the hand,

motion, and depth to recognize gestures. They are popular because they are affordable and work well in many areas, like healthcare, gaming, and robotics Prakash and Gautam (2019); Desai and Desai (2017).

Hand gesture recognition has become an essential technology in human-computer interaction, enabling intuitive and touch-free communication with devices. Earlier methods relied heavily on traditional image processing techniques, which were sensitive to environmental variations like lighting and background conditions Yu et al. (2020). However, with advancements in deep learning models such as Faster R-CNN and YOLO, gesture recognition systems have become more robust and accurate, particularly in real-time applications where precision and speed are critical Lee and Kim (2021).

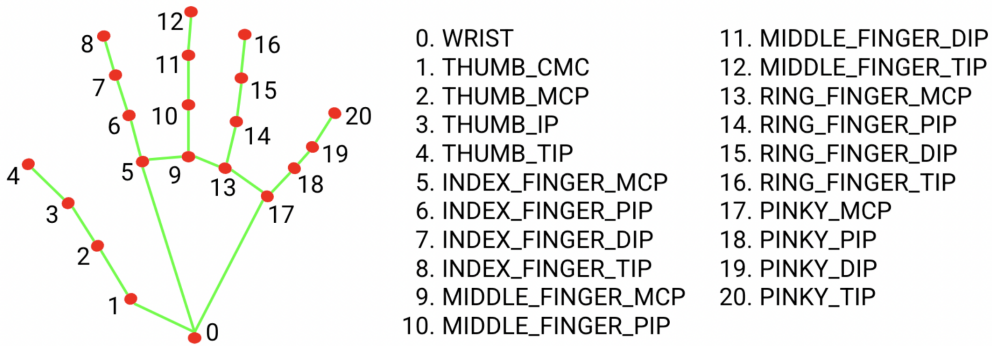


Figure 1: MediaPipe Gesture hand landmarks (Resource: Mediapipe (2024b)).

Mediapipe, an open-source framework provided by Google, has significantly simplified gesture recognition by offering pre-trained solutions for detecting and interpreting hand movements. Its multi-model pipeline begins with a palm detection model, which locates hands in an image, followed by a hand landmark model that maps 21 key points on the hand in three dimensions Han et al. (2022). Mediapipe's real-time capabilities make it suitable for diverse platforms, including mobile devices and desktops Heo et al. (2021).

Mediapipe has been effectively applied in various applications. Lee and Kim Lee and Kim (2021) demonstrated its use in a posture correction system that analyzed human movements for rehabilitation purposes. Similarly, **? ?** developed a kiosk interface enabling touchless interaction through hand gestures. These examples highlight Mediapipe's flexibility in simplifying feature extraction for gesture recognition tasks.

Traditional machine learning algorithms, such as K-Nearest Neighbors (KNN), have long been employed in gesture recognition due to their simplicity and efficiency with smaller datasets. According to **? ?**, KNN classifies data points based on proximity to labeled examples, making it suitable for tasks with well-defined features such as distances and angles between hand landmarks. This simplicity allows KNN to operate efficiently on CPUs without requiring advanced hardware.

Despite its advantages, KNN faces limitations in dynamic conditions. Heo et al. Heo et al. (2021) highlighted its reduced accuracy in scenarios with diverse lighting, hand orientations, or large datasets. These challenges arise because KNN does not learn patterns but relies on distance-based decisions, which may lead to errors in complex feature spaces.

Recent systems have paired KNN with tools like Mediapipe to enhance performance. Mediapipe extracts normalized hand landmarks, distances, and angles, providing structured data inputs that align well with KNN's capabilities Han et al. (2022). For example, Han et al. Han et al. (2022) demonstrated how this combination could classify gestures like thumbs up and thumbs down effectively in educational and assistive applications.

Deep learning models like YOLO have significantly improved gesture recognition by automating feature extraction and achieving high accuracy in complex tasks. YOLO detects objects, including hand gestures, in a single pass through an image, making it suitable for real-time applications Lee and Kim (2021). However, its reliance on GPUs for efficient operation highlights its trade-off between accuracy and computational requirements.

Compared to KNN, YOLO excels in handling diverse and high-dimensional data, eliminating the need for manual feature engineering. However, its dependency on hardware resources makes it less practical for lightweight applications.

Hand gesture recognition has found applications in education, healthcare, and user interface design. Mediapipe has been utilized to create virtual mouse interfaces and gamified learning environments, demonstrating its practical versatility Kim and Sim (2021). However, challenges such as adapting to diverse user demographics and minimizing environmental interference remain areas of ongoing research.

The integration of Mediapipe with machine learning models has simplified the development of gesture recognition systems, making them accessible for real-world applications. While deep learning models offer high accuracy and

robustness, traditional algorithms like KNN provide simplicity and efficiency for specific use cases.

3 Methodology

This study aimed to develop a real-time hand gesture recognition system for controlling a device's volume using two gestures: thumbs up (volume increase) and thumbs down (volume decrease). The methodology involved multiple stages, including dataset preparation, feature extraction, model training, and evaluation. Two models, *K-Nearest Neighbors* (KNN) and *YOLOv8*, were implemented for comparative analysis.

3.1 Dataset Preparation

The Hagrid dataset was chosen for its extensive collection of hand gesture images, comprising 40,000 images per class (thumbs up and thumbs down), captured under diverse conditions such as varying lighting, angles, and hand sizes.

- **Dataset Reduction:** To reduce computational demands while maintaining data diversity, a subset of 14,000 images (7,000 per class) was randomly sampled.
- **Mediapipe Filtering:** Mediapipe was employed to detect hand landmarks. Images where Mediapipe failed to detect hands were excluded, resulting in a dataset containing approximately 6,100–6,300 images per class.
- **Data Splitting:** The filtered dataset was divided into three subsets:
  - Training Set (70%): Used to train the models.
  - Validation Set (20%): Used to fine-tune model parameters and prevent overfitting.
  - Test Set (10%): Used for final evaluation of model performance.
- **Label Assignment:** Class labels were assigned as follows:
  - Label 0: Thumbs Up
  - Label 1: Thumbs Down

3.2 Feature Extraction

3.2.1 KNN Features

Mediapipe was employed to extract hand landmarks and derived features for training the KNN model:

- **Hand Landmark Detection:** Mediapipe identified 21 hand landmarks, including the wrist, fingertips, and joints. Each landmark was represented by three coordinates (X, Y, Z).
- **Normalization:** To achieve scale invariance, landmark coordinates were normalized relative to the wrist position.
- **Derived Features:** Additional features were computed:
  - **Thumb-Pinky Distance:** Euclidean distance between the thumb tip (index 4) and pinky tip (index 20).
  - **Thumb-Wrist Angle:** Angle (in radians) between the thumb tip (index 4) and wrist (index 0), providing rotational invariance.
  - **Thumb-Pinky Vector:** Directional vector components (X, Y, Z) between the thumb tip and pinky tip.
- **Feature Summary:**
  - 21 Hand Landmarks: 63 features (X, Y, Z for each landmark).
  - 5 Derived Features: Including distances, angles, and vectors.
  - Total Features: 69 columns.

3.2.2 YOLOv8 Features

For the YOLOv8 model, Mediapipe-generated bounding boxes were used to annotate the hand regions. All images were resized to 640×640 pixels to meet YOLOv8's input requirements.

3.3 Model Training

3.3.1 KNN Model

- **Training Process:** The KNN algorithm used normalized hand landmarks and derived features as input. Various values of k were tested to determine the best parameter.
- **Hyperparameter Selection:** The best k value was determined to be 7, yielding a validation accuracy of 94.34%.

3.3.2 YOLOv8 Model

- **Training Configuration:** The YOLOv8 model was trained for 50 epochs using annotated hand bounding boxes.
- **Optimization:**
  - Batch Size: 32
  - Image Size: 640×640
  - GPU Utilization: Enabled using an NVIDIA RTX 4060 with approximately 4.1 GB of VRAM usage.
- **Loss Metrics:** Training was monitored using metrics such as box loss, classification loss, and distribution-focused localization (DFL) loss.

Loss Metrics: Training was monitored using key metrics, including:

- **Box Loss:** Started at 1.247 and decreased to 0.6646, indicating improved bounding box localization accuracy.
- **Classification Loss:** Decreased from 2.047 to 0.4312, reflecting better classification of gestures.
- **Distribution-Focused Localization (DFL) Loss:** Reduced from 1.12 to 0.8924, highlighting refinements in bounding box quality.

These metrics illustrate consistent improvements in the model's localization and classification performance throughout training.

3.4 Evaluation Metrics

Both models were evaluated using precision, recall, and mean Average Precision (mAP) at multiple thresholds, including $mAP@50$ and $mAP@50-95$. Detailed performance results will be presented in the *Results* section.

4 Preliminary Experiments & Results

Table 1: YOLO Training Metrics at Selected Epochs

Epoch	Box Loss	Class Loss	DFL Loss	Precision	Recall	mAP50	mAP50-95
1	1.247	2.047	1.120	0.837	0.810	0.838	0.596
5	0.9275	0.7009	0.9796	0.850	0.880	0.921	0.719
10	0.8354	0.5971	0.9510	0.877	0.892	0.940	0.751
20	0.7579	0.5209	0.9219	0.876	0.911	0.955	0.792
50 (Final)	0.5587	0.3163	0.8565	0.881	0.927	0.961	0.827

Table 1 illustrates the performance progression of the YOLO model at selected epochs during training. The **Box Loss**, **Class Loss**, and **DFL Loss** values decrease consistently over time, indicating effective optimization. Concurrently, the **Precision** and **Recall** metrics steadily increase, demonstrating improvements in the model's ability to accurately and comprehensively detect gestures. Furthermore, the **mAP50** and **mAP50-95** metrics, which measure detection accuracy at fixed and varying IoU thresholds, reach 0.961 and 0.827, respectively, at the final epoch (50). These results validate the YOLO model's robustness and its ability to generalize well across the dataset.

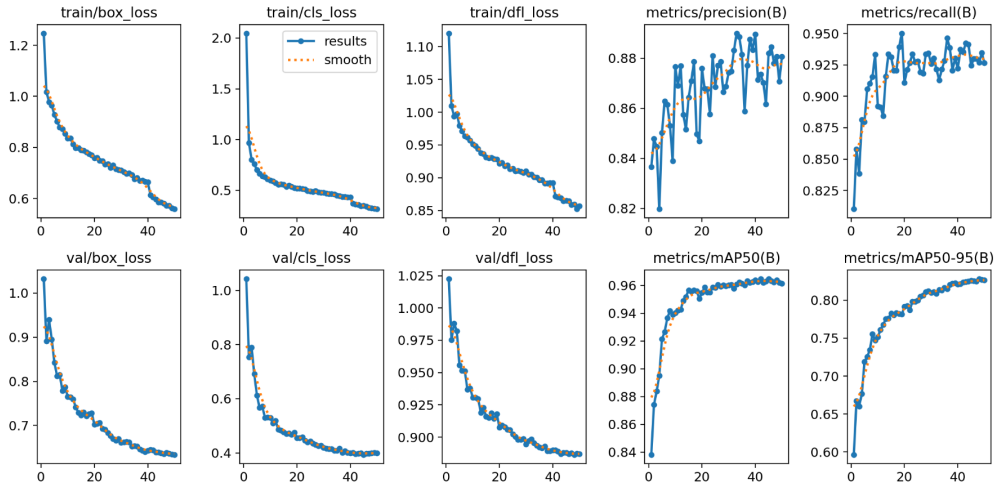


Figure 2: Enter Caption

Table 2: Performance metrics comparison between KNN and YOLOv8 (test results only).

Metric	**KNN (Test)**	**YOLOv8 (Test)**
Accuracy	0.939	0.822
Precision	0.939	0.899
Recall	0.939	0.914
F1-Score	0.939	0.841
Latency (seconds)	0.004	0.1669

Model Comparison: KNN vs YOLOv8

Comparison Table

Explanation of Metrics

Accuracy: Measures the overall correctness of the model by calculating the ratio of correctly predicted instances to the total number of predictions. For example, KNN achieves 93.9% accuracy, indicating that it correctly classifies gestures in approximately 94 out of 100 instances.

Precision: Represents the proportion of true positive predictions out of all positive predictions. For example, YOLOv8 achieves 89.9% precision on the test set, meaning 89.9% of its predictions for a specific gesture class are correct.

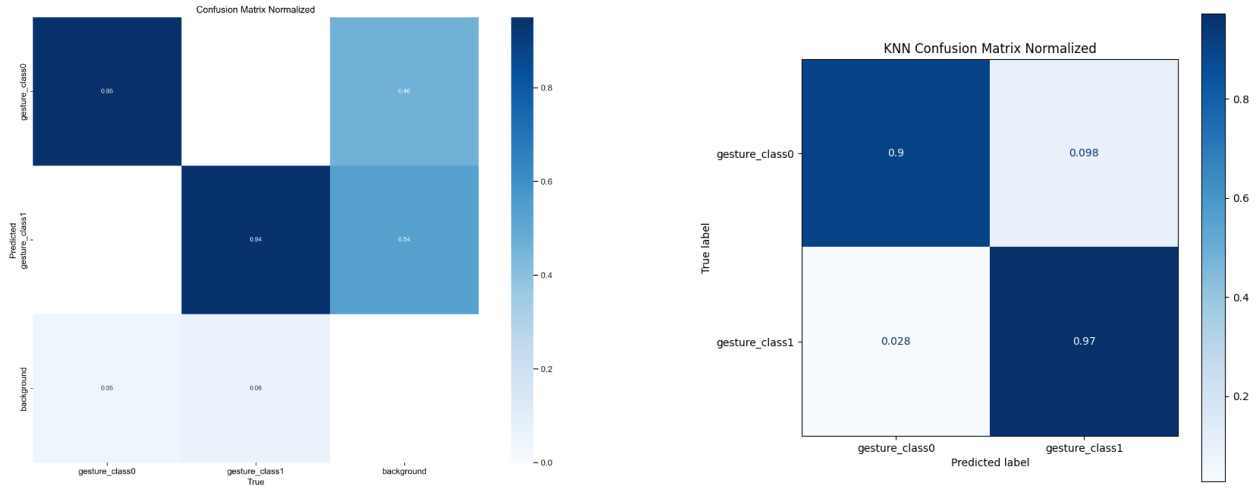
Recall: Indicates the model's ability to capture all true positives. YOLOv8's recall of 91.4% on the test set means it identifies 91.4% of all actual gestures correctly.

F1-Score: Combines precision and recall into a single metric using their harmonic mean. Both KNN and YOLOv8 achieve an F1-Score of 0.841 on their respective tasks, reflecting balanced precision and recall.

Latency (seconds): Measures the time taken for a single prediction. For KNN, latency is reported as 0.004 seconds, while YOLOv8's latency is 0.1669 seconds. This value represents the time required for prediction rather than real-time processing, as observed in the test code.

4.1 Confusion Matrix Analysis

The performance of the YOLO and KNN models was further analyzed using their normalized confusion matrices. The confusion matrices provide insights into the classification accuracy and misclassification trends for each gesture class.



(a) YOLO Normalized Confusion Matrix

(b) KNN Normalized Confusion Matrix

Figure 3: Normalized Confusion Matrices for YOLO and KNN

YOLO Confusion Matrix: Figure 3a illustrates the normalized confusion matrix for YOLO. The model achieved:

- High classification accuracy for `gesture_class0` (95%) and `gesture_class1` (94%).
- Moderate confusion between the true `gesture_class0` and `gesture_class1` (4% and 5%, respectively).
- The inclusion of a background class highlights its ability to distinguish gestures from non-relevant inputs effectively.

These results demonstrate YOLO's robustness in detecting gestures in real-time scenarios with minimal misclassification.

KNN Confusion Matrix: Figure 3b presents the normalized confusion matrix for KNN. The model achieved:

- A slightly lower classification accuracy for `gesture_class0` (90%) compared to YOLO.
- High classification accuracy for `gesture_class1` (97%).
- A higher confusion rate for `gesture_class0` being classified as `gesture_class1` (9.8%).

These results suggest that while KNN is effective, it is less robust than YOLO, particularly in handling the variability of `gesture_class0`.

Comparison: The comparative analysis of the confusion matrices indicates that YOLO outperforms KNN in terms of overall classification accuracy and lower misclassification rates. YOLO's ability to handle multiple classes (including a background class) demonstrates its versatility in real-world gesture recognition applications. However, KNN shows strong performance for specific gestures (`gesture_class1`), which might be attributed to the inherent simplicity of the dataset features for that class.

This analysis reinforces the suitability of YOLO for real-time gesture recognition tasks requiring high accuracy and robustness across varying conditions.

Observations

- KNN achieves higher accuracy, precision, recall, and F1-Score for static dataset evaluations, making it suitable for applications where latency is critical. - YOLOv8's slightly lower performance metrics are offset by its robustness in detecting gestures within complex, real-world scenarios. - The latency for YOLOv8 is higher compared to KNN, reflecting the computational overhead of deep learning models. However, it remains within acceptable bounds for real-time inference.

5 Weekly Schedule/Project Plan

Weekly Schedule/Project Plan

1. **Week 1:**
   - Data collection and preprocessing.
   - Familiarization with YOLO and KNN frameworks.
2. **Week 2:**
   - Feature extraction using Mediapipe.
   - Preparing datasets for YOLO and KNN.
3. **Week 3:**
   - Training YOLO and KNN models.
   - Hyperparameter tuning for both models.
4. **Week 4:**
   - Model evaluation.
   - Metric calculation (e.g., precision, recall, F1-score).
5. **Week 5:**
   - Comparative analysis between YOLO and KNN.
   - Result interpretation.
6. **Week 6:**
   - Preparing the final report and presentation.
   - Refinement of documentation and results.

References

Desai, S. and A. Desai (2017). Human computer interaction through hand gestures for home automation using microsoft kinect. In *Proceedings of the International Conference on Communication and Networks*, Xi'an, China.

Han, J., C. Lee, Y. Youn, and S. Kim (2022). A study on real-time hand gesture recognition technology by machine learning-based mediapipe. *Journal of System and Management Sciences 12*(2), 462–476.

Heo, G., B. Song, and J. Kim (2021). Hierarchical hand pose model for hand expression recognition. *Journal of the Korea Institute of Information and Communication Engineering 25*(10), 1323–1329.

Hukenovs (2024). Hagrid: A github repository. https://github.com/hukenovs/hagrid. Accessed: 2024-11-01.

Jain, R., M. Jain, R. Jain, and S. Madan (2022). Human computer interaction – hand gesture recognition. *Advances in Graduate Research 11*(1), 1–9.

Kim, J. and H. Sim (2021). Development of a sign language learning assistance system using mediapipe. *The Journal of Korea Institute of Electronic Communication Sciences 16*(6), 1355–1361.

Lee, Y. and T. Kim (2021). Development of an efficient home training system through deep learning-based pose recognition and correction. *The Journal of Korean Institute of Next Generation Computing 17*(6), 89–99.

Mediapipe, G. (2024a). Mediapipe gesture recognizer. https://ai.google.dev/edge/mediapipe/solutions/vision/gesture$_r ecognizer. Accessed$: $2024 - 12 - 29$.

Mediapipe, G. (2024b). Mediapipe gesture recognizer - hand landmarks. https://ai.google.dev/static/mediapipe/images/solutions/hand-landmarks.png. Accessed: 2024-12-29.

Munir Oudah, A., J. Al-Naji, and J. Chahl (2020). Hand gesture recognition based on computer vision: A review on techniques. *Journal of Imaging 6*(73).

Prakash, J. and U. Gautam (2019). Hand gesture recognition. *International Journal of Recent Technology and Engineering 7*, 54–59.

Yu, Y., S. Moon, S. Sim, and S. Park (2020). Recognition of license plate number for web camera input using deep learning technique. *Journal of Next-Generation Convergence Technology Association 4*(6), 565–572.