Software Engineering

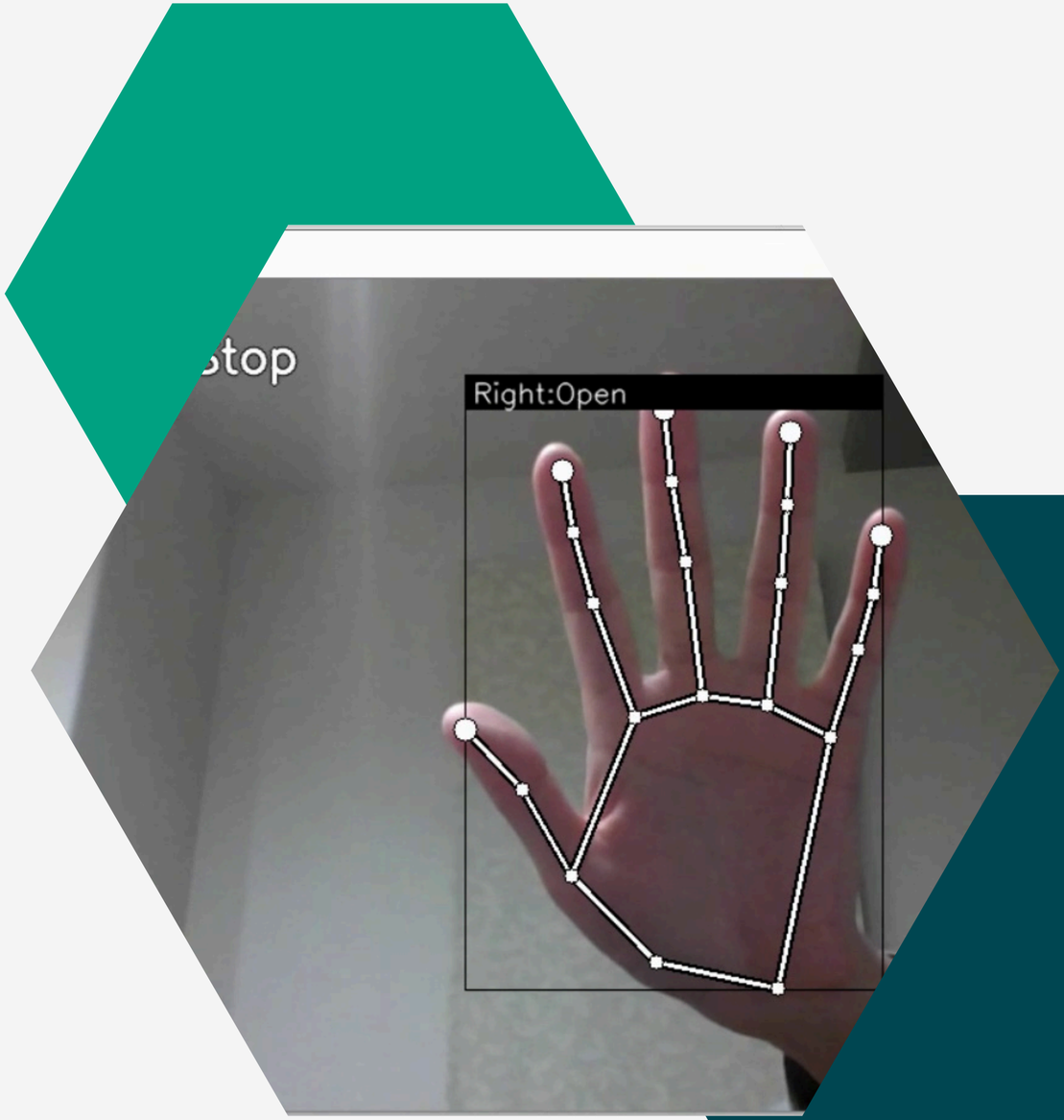
Hand Gesture Recoginition

Agenda

- Introduction
- Overview of Lifecycle
- Discovery
- Data Preparation
- Model Planning
- Model Building
- Communication of Results
- Operationalize
- Conclusion

Introduction

- Overview of the project: Hand gesture recognition using MediaPipe is a cutting-edge technology that allows computers to interpret human hand gestures through a webcam feed.
- Importance of hand gesture recognition: Hand gesture recognition has numerous applications, including sign language translation, human-computer interaction, virtual reality, and augmented reality.
- Use cases and applications: Examples include real-time sign language translation for the deaf community, gesture-based control in video games, and interactive interfaces in smart devices.



Lifecycle

Initiation Phase:

Project Identification:
Stakeholder Analysis:
Project Charter:

Planning Phase:

Requirement Gathering:
Resource Planning:
Risk Management:
Project Schedule:

Execution Phase:

Data Collection:
Model Development:
Training and Testing:
System Integration:

Monitoring and Control Phase:

Performance Monitoring:
Quality Assurance:
Change Management:
Communication:

Closure Phase:

Project Evaluation:
Documentation:
Knowledge Transfer:
Celebration and Reflection:

Initiation Phase:

Problem statement and Domain

The problem we aim to address is the need for efficient and accurate hand gesture recognition systems across various domains, including human-computer interaction, virtual reality, robotics, and assistive technology. Current systems often face challenges such as limited recognition accuracy, high computational complexity, and lack of robustness to diverse environmental conditions and user variations. Our goal is to develop a robust and real-time hand gesture recognition system that can accurately interpret a wide range of hand gestures with minimal latency and resource requirements.

Stakeholder Analysis:

Stakeholders include researchers in the fields of computer vision and machine learning, software developers, end-users (such as individuals with disabilities or professionals in industries like gaming and healthcare), and potential investors or collaborators.

Project Charter:

he project charter outlines the scope of the project, which includes developing a prototype hand gesture recognition system, conducting performance evaluation tests, and exploring potential applications and commercialization opportunities.

Data Preparation

Data Collection:

- Identify existing hand gesture datasets available in the public domain, such as American Sign Language (ASL) datasets or gesture recognition datasets from academic sources.
- If necessary, design and conduct custom data collection sessions to capture a diverse range of hand gestures using RGB or depth sensors.
- Ensure that the collected data cover various hand shapes, orientations, lighting conditions, and backgrounds to improve the model's robustness.

Data Preparation

Data Cleaning and Annotation:

- Remove any irrelevant or noisy data samples from the collected dataset.
- Standardize the data format and labeling conventions to ensure consistency across the dataset.
- Annotate each data sample with the corresponding hand gesture label, either manually or using automated annotation tools.

Data Augmentation:

- Augment the dataset by applying transformations such as rotation, scaling, translation, and flipping to increase the diversity of training examples.
- Introduce synthetic variations in lighting conditions, background clutter, and occlusions to simulate real-world scenarios and improve the model's generalization capabilities.

Data Splitting:

- Divide the annotated dataset into training, validation, and test sets to facilitate model training, evaluation, and performance validation.
- Ensure that each set contains a representative distribution of hand gestures to avoid bias and overfitting during model training and evaluation.

Data Preparation

Data Preprocessing:

- Normalize the input data to ensure consistency in feature scales and ranges, which helps stabilize the training process and improve model convergence.
- Apply preprocessing techniques such as image resizing, cropping, and color normalization to standardize the input data format and reduce computational complexity.
- Perform feature extraction if necessary, such as extracting hand keypoints or descriptors using computer vision algorithms, to represent the input data effectively for model training.

Data Pipeline Setup:

- Develop data loading and preprocessing pipelines using appropriate programming libraries (e.g., OpenCV, TensorFlow, PyTorch) to efficiently handle large-scale datasets and streamline model training workflows.
- Implement data augmentation and batch processing techniques to enhance data diversity and optimize computational resources during model training.

Data Quality Assurance:

- Conduct thorough data quality checks and validation procedures to identify and address any inconsistencies, errors, or biases in the dataset.
- Validate the correctness of data annotations and labels through manual inspection or automated verification methods to ensure the reliability of the training data.

Data Preparation

```
4] 1 X_dataset = np.loadtxt(dataset, delimiter=',', dtype='float32', usecols=list(range(1, (21 * 2) + 1)))

5] 1 y_dataset = np.loadtxt(dataset, delimiter=',', dtype='int32', usecols=(0))

6] 1 X_train, X_test, y_train, y_test = train_test_split(X_dataset, y_dataset, train_size=0.75, random_state=RANDOM_SEE
```

Add Code Cell | Add Markdown Cell

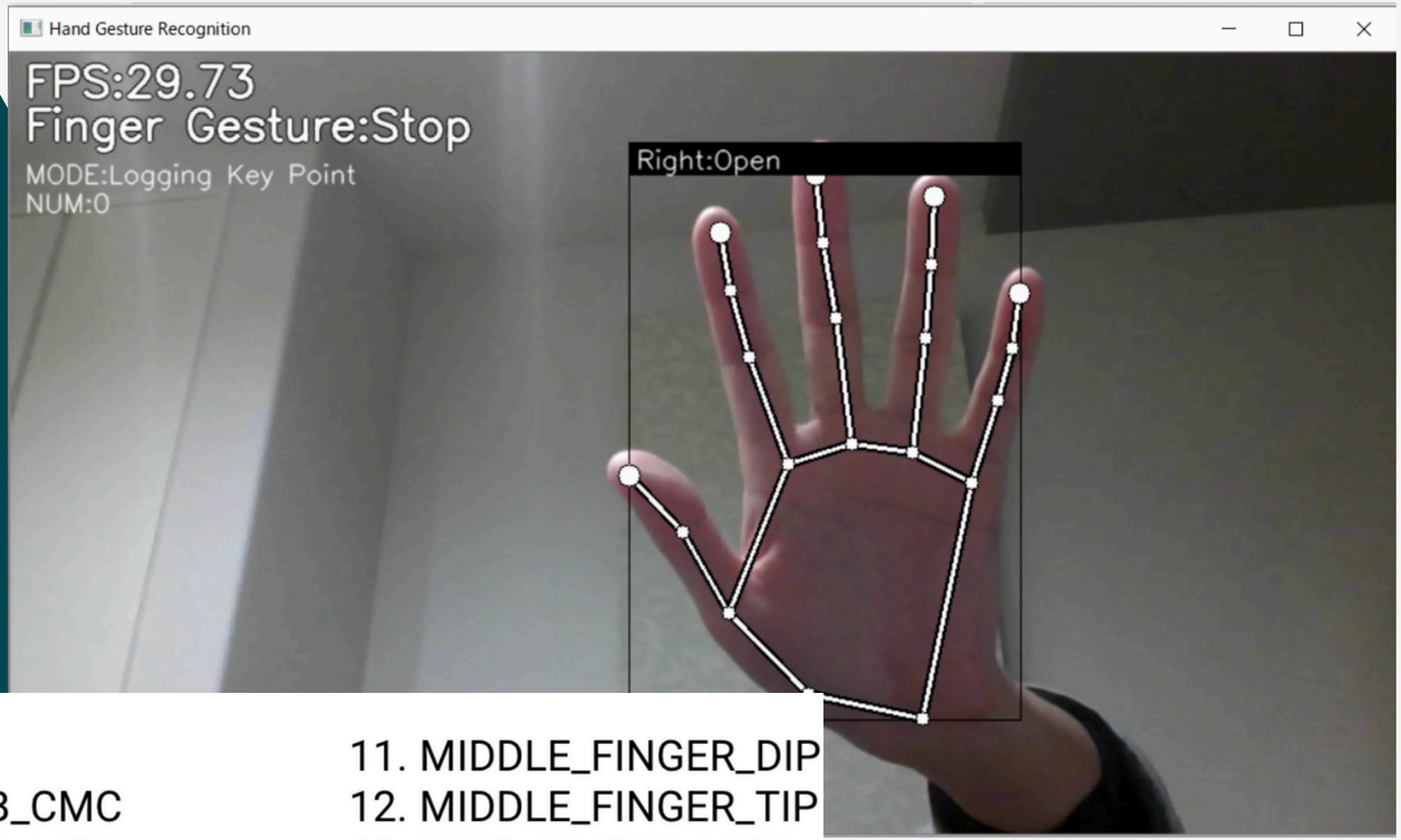
Model building

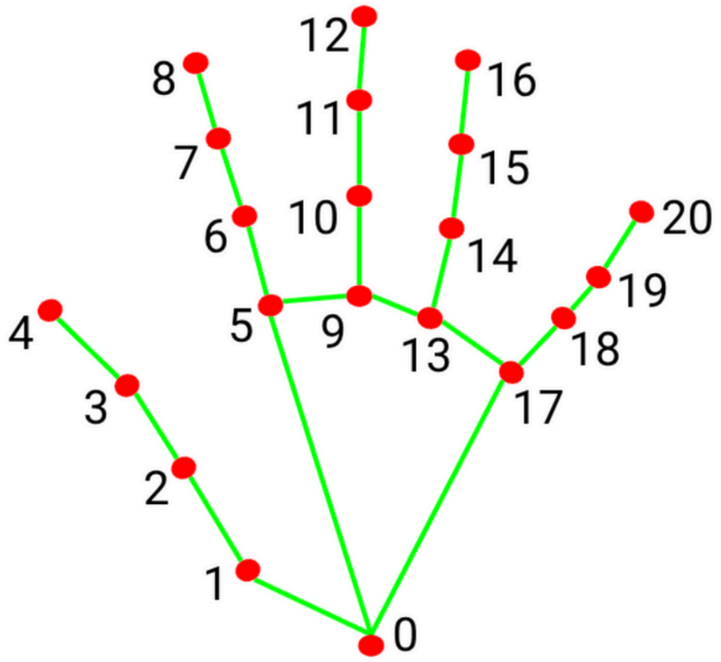
```
7] 1 model = tf.keras.models.Sequential([
   2     tf.keras.layers.Input((21 * 2, )),
   3     tf.keras.layers.Dropout(0.2),
   4     tf.keras.layers.Dense(20, activation='relu'),
   5     tf.keras.layers.Dropout(0.4),
   6     tf.keras.layers.Dense(10, activation='relu'),
   7     tf.keras.layers.Dense(NUM_CLASSES, activation='softmax')
   8 ])

8] 1 model.summary()  # tf.keras.utils.plot_model(model, show_shapes=True)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dropout (Dropout)	(None, 42)	0
dense (Dense)	(None, 20)	860
dropout_1 (Dropout)	(None, 20)	0
dense_1 (Dense)	(None, 10)	210

Data Preparation



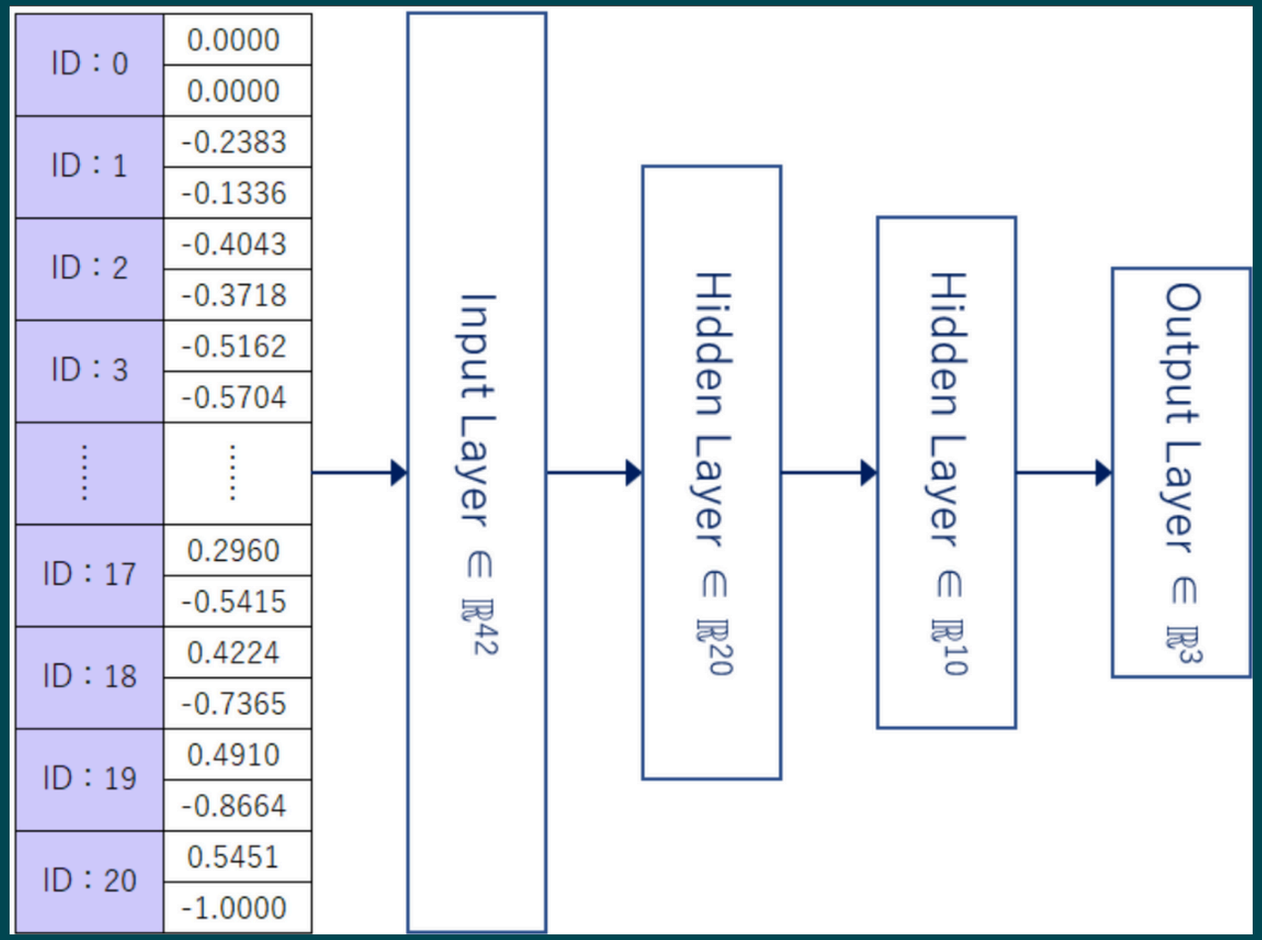


0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Visualization

- Data visualization is a powerful tool for interpreting and communicating complex data.
- It helps in uncovering patterns, trends, and insights that are not easily apparent from raw data alone.

Data Visualization



① 各キーポイント座標（Landmark coordinates）

ID：0	ID：1	ID：2	ID：3	……	ID：17	ID：18	ID：19	ID：20
[551, 465]	[485, 428]	[439, 362]	[408, 307]	……	[633, 315]	[668, 261]	[687, 225]	[702, 188]

② ID:0からの相対座標に変換（Convert to relative coordinates from ID:0）

ID：0	ID：1	ID：2	ID：3	……	ID：17	ID：18	ID：19	ID：20
[0, 0]	[-66, -37]	[-112, -103]	[-143, -158]	……	[82, -150]	[117, -204]	[136, -240]	[151, -277]

③ 1次元配列に変換（Flatten to a one-dimensional array）

ID：0		ID：1		ID：2		ID：3		……	ID：17		ID：18		ID：19		ID：20	
0	0	-66	-37	-112	-103	-143	-158	……	82	-150	117	-204	136	-240	151	-277

④ 最大値(絶対値)に合わせて正規化（Normalized to the maximum value(absolute value)）

ID：0		ID：1		ID：2		ID：3		……	ID：17		ID：18		ID：19		ID：20	
0	0	-0.24	-0.13	-0.4	-0.37	-0.52	-0.57	……	0.296	-0.54	0.422	-0.74	0.491	-0.87	0.545	-1

Model Planning and building

Model Planning and Building

Data Preparation:

- Data Collection: Gather diverse hand gesture datasets from reliable sources or capture new samples using compatible hardware devices.
- Data Annotation: Label each data sample with the corresponding hand gesture using standardized conventions and annotation tools.
- Data Preprocessing: Clean and standardize the dataset, resize images, normalize pixel values, and address class imbalance issues.

Model Architecture:

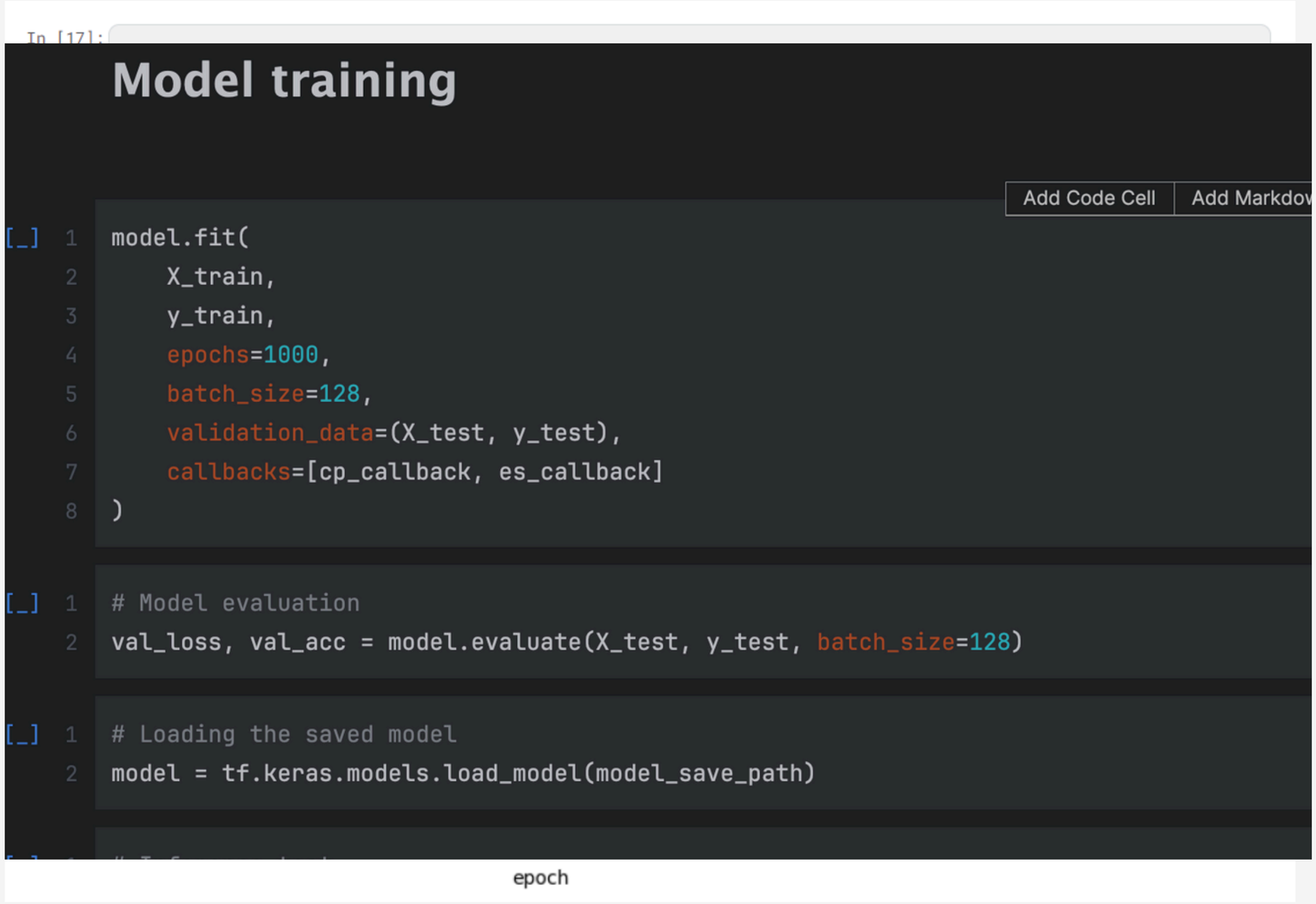
- Convolutional Neural Network (CNN) architecture is typically employed. CNNs are adept at capturing spatial features from image data, making them suitable for tasks like recognizing hand gestures. The architecture typically consists of convolutional layers for feature extraction, activation functions for introducing non-linearity, pooling layers for downsampling, fully connected layers for high-level representation learning, and an output layer for producing predictions. By customizing these components and optimizing hyperparameters, we can build an effective model for hand gesture recognition.

Model Planning and Building

Model Training:

- Data Preprocessing: Prepare the dataset by resizing images, normalizing pixel values, and splitting into training, validation, and testing sets.
- Model Definition: Design the architecture of the CNN model, specifying the number and configuration of layers, activation functions, dropout layers, etc.
- Compile the Model: Choose the appropriate loss function (e.g., categorical cross-entropy for classification tasks), optimizer (e.g., Adam, SGD), and evaluation metrics (e.g., accuracy).
- Training: Feed the training data into the model and adjust the weights iteratively through backpropagation to minimize the loss function. This involves epochs and batches, where an epoch is one pass through the entire training dataset, and a batch is a subset of the training data used in one iteration.
- Validation: After each epoch, evaluate the model's performance on the validation set to monitor for overfitting and adjust hyperparameters accordingly.
- Hyperparameter Tuning: Fine-tune hyperparameters such as learning rate, batch size, and architecture configurations to optimize the model's performance.
- Testing: Once training is complete, evaluate the final model on the testing dataset to assess its generalization ability and performance metrics.
- Iterative Optimization: Based on the testing results, refine the model further if necessary by adjusting parameters, adding regularization techniques, or collecting more data.

Model Planning and Building



Model Planning and Building

Model building

[7]
```
model = tf.keras.models.Sequential([
    tf.keras.layers.Input((21 * 2, )),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(20, activation='relu'),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(NUM_CLASSES, activation='softmax')
])
```

[8]
```
model.summary()  # tf.keras.utils.plot_model(model, show_shapes=True)
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dropout (Dropout)	(None, 42)	0
dense (Dense)	(None, 20)	860
dropout_1 (Dropout)	(None, 20)	0
dense_1 (Dense)	(None, 10)	210
dense_2 (Dense)	(None, 5)	55

```
Total params: 1,125 (4.39 KB)

Trainable params: 1,125 (4.39 KB)

Non-trainable params: 0 (0.00 B)
```

Communication And Results

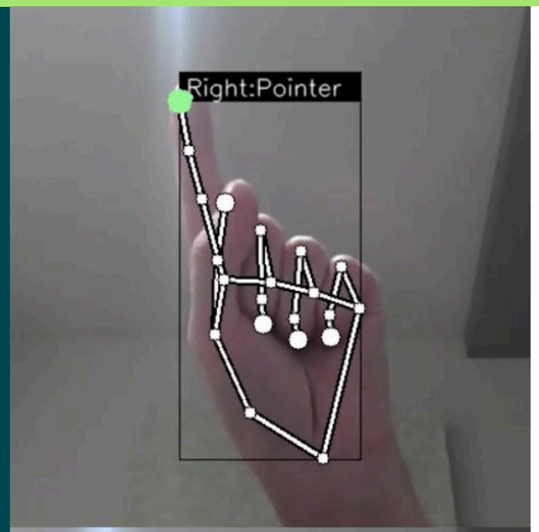
Communication And Results

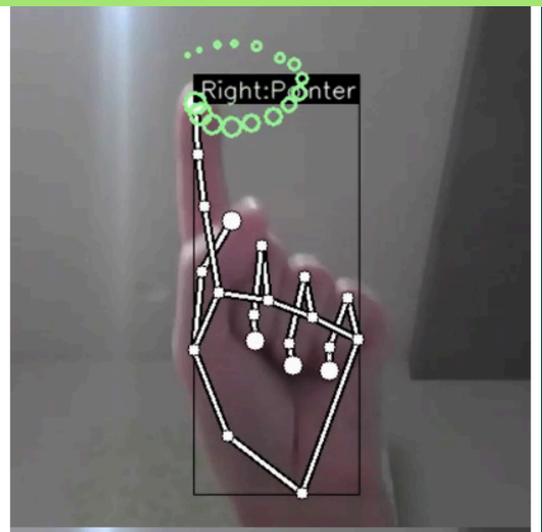
- The provided project aims to recognize hand signs and finger gestures using MediaPipe in Python.
- It includes:
- Sample program for inference.
- Hand sign recognition model (TFLite).
- Finger gesture recognition model (TFLite).
- Training data and notebooks for hand sign and finger gesture recognition.
- Requirements include mediapipe, OpenCV, TensorFlow, scikit-learn, and matplotlib.
- You can run the demo using your webcam with the provided command.
- Training instructions are provided for both hand sign and finger gesture recognition.
- The project structure, including directories and files, is clearly outlined.
- The provided Python script handles camera capture, hand detection, gesture recognition, and visualization of the detected landmarks and gestures in real-time.

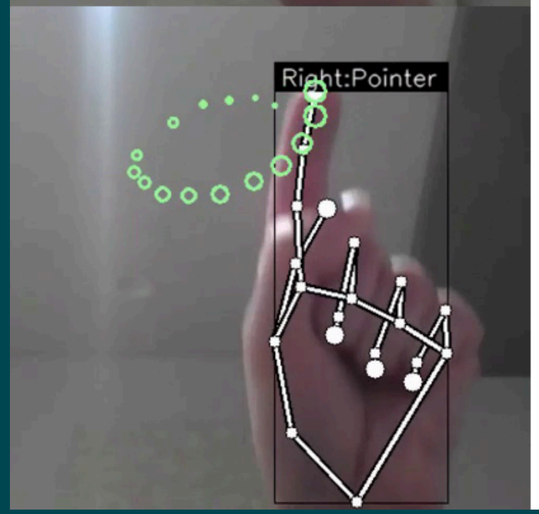
Communication And Results

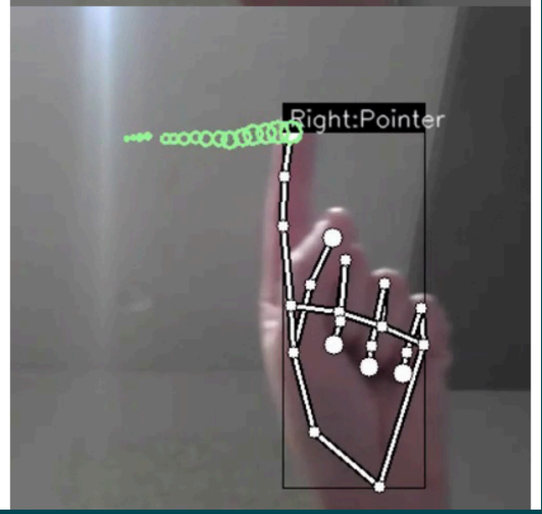
- Project Overview:
- The project is centered around hand gesture recognition using the MediaPipe library in Python.
- It comprises several components, including model training, inference, and real-time visualization.
- Key Components:
- Sample Program: You provided a sample Python program for inference using the trained models.
- Models: The project includes pre-trained models for both hand sign recognition and finger gesture recognition, implemented in TensorFlow Lite (TFLite) format.
- Training Data and Notebooks: You included training data and Jupyter notebooks for training the hand sign and finger gesture recognition models.
- Dependencies:
- The project relies on various Python libraries, including MediaPipe, OpenCV, TensorFlow, scikit-learn, and matplotlib.
- Demo Instructions:
- You outlined instructions for running the demo using a webcam, specifying the command to execute the provided Python script.
- Training Instructions:
- Detailed instructions were provided for training both the hand sign and finger gesture recognition models, leveraging the provided data and notebooks.
- Project Structure:
- The directory structure of the project was clearly outlined, facilitating easy navigation and understanding of the codebase.
- Functionality:
- The Python script included functionality for camera capture, hand detection, gesture recognition, and real-time visualization of detected landmarks and gestures.

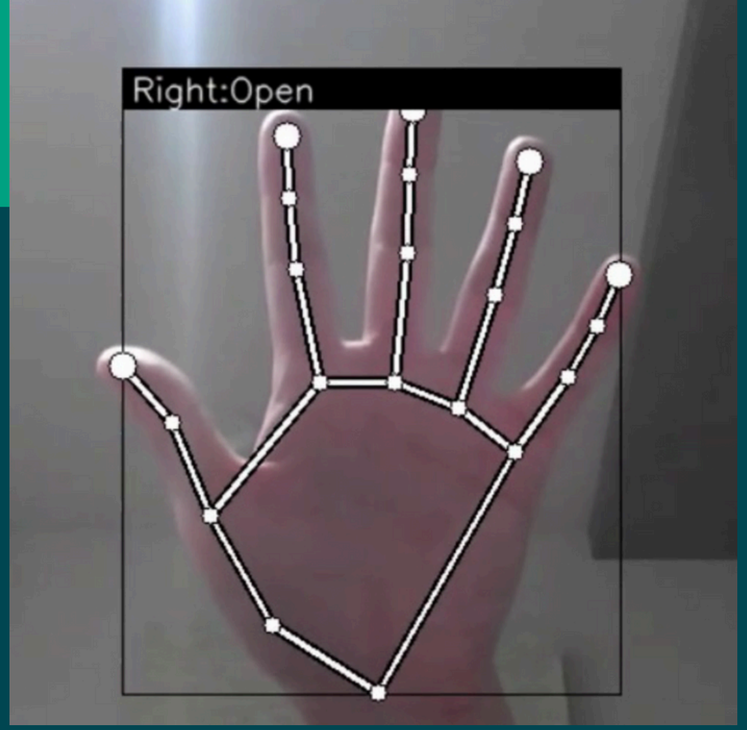
Sample Predictions

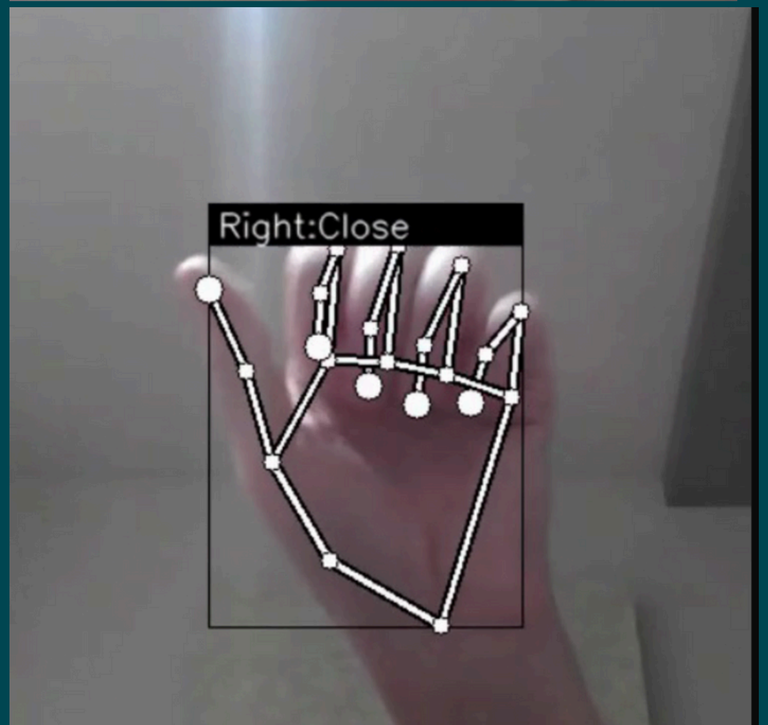












Operationalize

Operationalize

- To operationalize the hand gesture recognition system using MediaPipe:
- Setup: Install required dependencies and clone the repository.
- Training: Train hand sign and finger gesture recognition models using provided Jupyter Notebooks.
- Data Collection: Use app.py to collect and label data for gestures.
- Inference: Run app.py to perform real-time gesture recognition from webcam feed.
- Visualization: View webcam feed with overlaid hand landmarks and recognized gestures.
- Interpretation: Analyze recognition results from displayed information.
- Integration: Integrate system into desired application or pipeline.
- Deployment: Deploy system on target platform after optimization and testing.

Conclusion

Conclusion

The hand-gesture-recognition-using-mediapipe repository offers pre-trained models and training data for recognizing hand signs and finger gestures. With clear instructions and demos, developers can easily deploy and customize these models for real-time applications, making it a valuable resource for hand gesture recognition projects.

Team Member

Shashank – 21BCE2221