NAME: COVENANT OLUDAPOMOLA OJO

COURSE: MACHINE LEARNING

STUDENT ID: 23013682

GITHUB LINK: **https://github.com/Covpet/DEEP-LEARNING-APPROACHES-FOR-REAL-TIME-AND-SCALABLE-SIGN-LANGUAGE-RECOGNITION-SYSTEMS/tree/main**

DATASET: **https://www.kaggle.com/datasets/ayuraj/asl-dataset/data**

DEEP LEARNING APPROACHES FOR REAL-TIME AND SCALABLE SIGN LANGUAGE RECOGNITION SYSTEMS

ABSTRACT

Sign language is a vital communication tool for individuals with hearing and speech impairments, enabling them to express thoughts and emotions through hand gestures and movements. However, the lack of universal understanding limits its accessibility in broader societal contexts. This project focuses on developing a scalable system for recognizing sign language gestures using Convolutional Neural Networks (CNNs). The system classifies hand gestures representing letters (A–Z) and numbers (0–9) by leveraging deep learning to automatically extract and learn features from gesture images. The model was trained on a structured dataset and achieved a high accuracy of 96% on the training set and 94% on the validation set, demonstrating its effectiveness. This system provides a practical, automated approach to bridging the communication gap between the hearing-impaired community and the wider population.

1. INTRODUCTION

Communication plays a crucial role in human interaction. For individuals with hearing and speech impairments, sign language is their primary means of communication. By using hand gestures, facial expressions, and body movements, they can express their thoughts and emotions. However, the limited understanding of sign language among the general population and the absence of a universal standard creates significant barriers. This gap restricts effective communication between the hearing-impaired community and society at large.

Advancements in artificial intelligence (AI) and machine learning have paved the way for innovative solutions to address these challenges. Convolutional Neural Networks (CNNs), a specialized deep learning technique, excel at analyzing visual data such as images. This project leverages CNNs to recognize sign language gestures, enabling the classification of alphabets and numbers with high accuracy. This report details the dataset, preprocessing steps, CNN architecture, training process, results, and challenges encountered, providing a comprehensive guide to implementing a sign language recognition system.

2. LITERATURE SURVEY

Research in sign language recognition has evolved significantly with the integration of machine learning and deep learning techniques.

2.1 TRADITIONAL METHODS

Earlier approaches relied on classical machine learning techniques like Support Vector Machines (SVMs) and k-Nearest Neighbors (k-NN), where features such as edges and shapes were manually extracted from images. While effective for small-scale datasets, these methods struggled with accuracy and scalability in real-world scenarios (Viswanathan et al., 2023).

2.2 ADOPTION OF CNNS

The introduction of Convolutional Neural Networks (CNNs) addressed the limitations of manual feature extraction. CNNs automatically learn patterns directly from images, enabling higher accuracy and better generalization. For instance, developed a CNN-based system that significantly improved recognition performance compared to traditional models. Raut et al. (2023)

2.3. CHALLENGES AND IMPROVEMENTS

Despite these advancements, challenges persist, such as the need for larger datasets and handling gestures with subtle differences. Techniques like data augmentation and transfer learning have been employed to enhance model performance, but real-world conditions such as varying lighting and hand orientations remain hurdles (Jim et al., 2024).

2.4 WHAT IS A CNN?

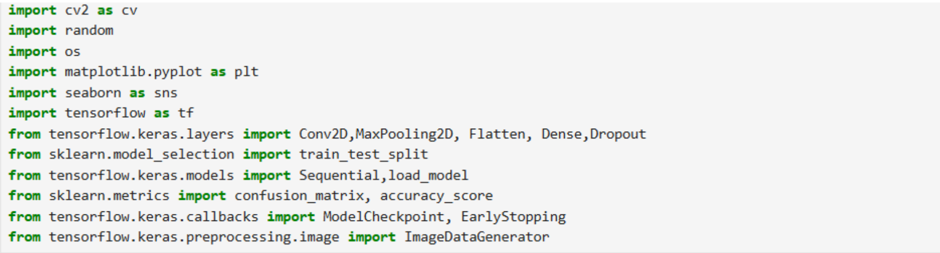
A Convolutional Neural Network (CNN) is a specialized type of deep learning model primarily designed to process and analyze visual data, such as images or video frames. Unlike traditional machine learning models that require manually defined features, CNNs are capable of automatically learning and extracting relevant features from images. This makes them particularly effective in tasks such as object detection, facial recognition, and image classification, where complex visual patterns need to be understood.(Viswanathan et al., 2023)

At a prominent level, CNNs consist of multiple layers, each with a specific function:

- **Convolutional Layers**: These layers apply filters (or kernels) that scan over the input image to detect various features, such as edges, textures, and shapes. Filters are small grids of numbers that "slide" over the image to extract local patterns.
- **Pooling Layers**: After convolutional layers, pooling layers reduce the spatial dimensions of the feature maps. This makes the network more efficient by minimizing the computational load, while retaining the most key features.
- **Fully Connected Layers**: These layers process the extracted features and combine them to make final predictions, typically in the form of class probabilities.

Activation Functions:

- **ReLU (Rectified Linear Unit):** Introduces non-linearity, allowing the model to learn complex patterns.
- **Softmax:** Converts the final output into class probabilities.



```
import cv2 as cv
import random
import os
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.keras.layers import Conv2D,MaxPooling2D, Flatten, Dense,Dropout
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential,load_model
from sklearn.metrics import confusion_matrix, accuracy_score
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Figure 1: importation of libraries (CNN component)

3.1 DATASET OVERVIEW

The dataset used in this project consists of images representing hand gestures for thirty-six classes: ten numeric digits (0–9) and twenty-six alphabetic characters (A–Z). Each class contains training and validation samples, ensuring a balanced representation for effective model learning.

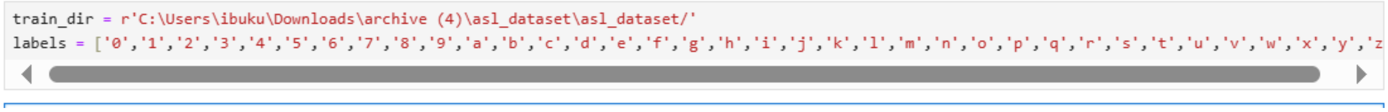
Key features of the dataset:

Classes: 36 (alphabets and numbers).

Training samples: sixty images per class.

Validation samples: ten images per class.

The images vary in resolution and orientation, requiring preprocessing to ensure uniformity and compatibility with the CNN model.



```
train_dir = r'C:\Users\ibuku\Downloads\archive (4)\asl_dataset\asl_dataset/'
labels = ['0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z
```

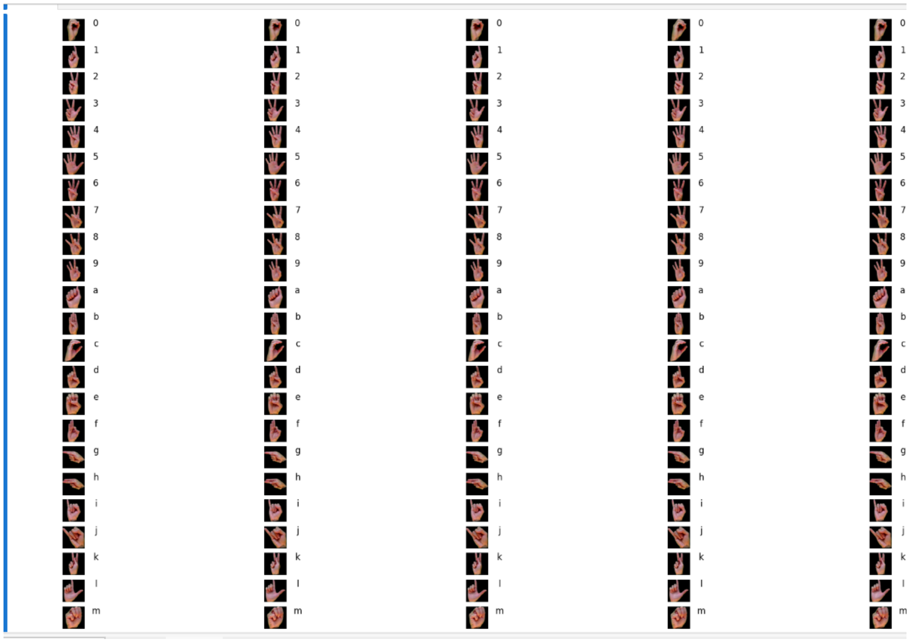


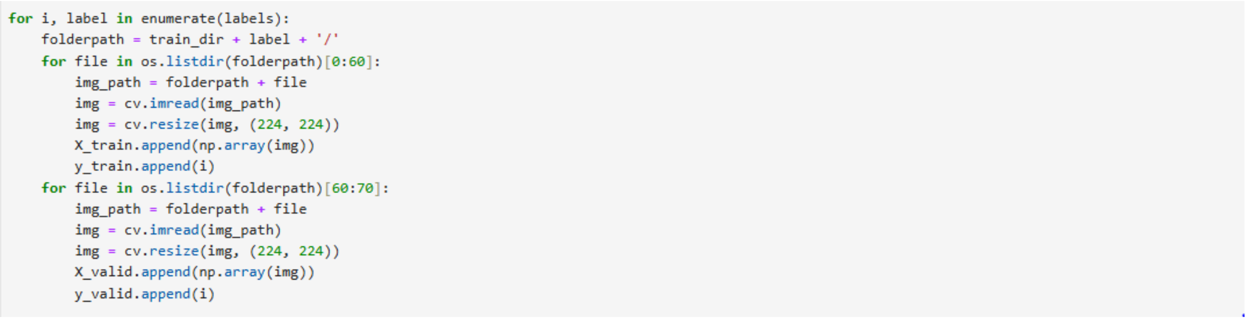
Figure 2: The labeled dataset (0-9, A-Z)

3.2 PREPROCESSING STEPS

To prepare the dataset for training, several preprocessing steps are applied to ensure that the images are consistent and suitable for feeding into the CNN:

- **Resizing**: Images are resized to 224x224 pixels to standardize their dimensions and make them suitable for processing by the CNN.
- **Normalization**: Pixel values, which initially range from 0 to 255, are scaled to a range of [0, 1] to improve the stability of the model during training.
- **Augmentation**: To artificially increase the size of the dataset and improve the model's ability to generalize, image augmentation techniques like rotation, flipping, and zooming

are applied. These techniques simulate variations in gesture orientations, helping the model learn to recognize gestures under different conditions.



```
for i, label in enumerate(labels):
    folderpath = train_dir + label + '/'
    for file in os.listdir(folderpath)[0:60]:
        img_path = folderpath + file
        img = cv.imread(img_path)
        img = cv.resize(img, (224, 224))
        X_train.append(np.array(img))
        y_train.append(i)
    for file in os.listdir(folderpath)[60:70]:
        img_path = folderpath + file
        img = cv.imread(img_path)
        img = cv.resize(img, (224, 224))
        X_valid.append(np.array(img))
        y_valid.append(i)
```

Figure 3: Image preprocessing

4. CNN ARCHITECTURE

The CNN model for sign language recognition consists of the following layers:

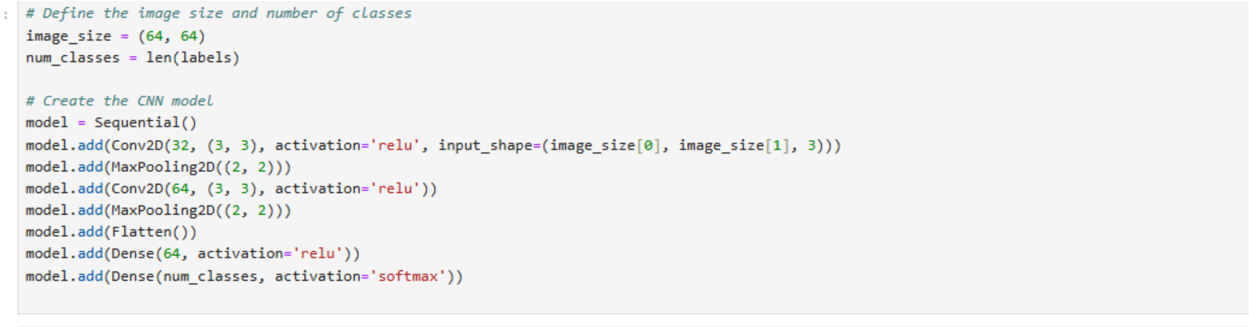
a. **Input Layer:** Accepts images of size 224x224 with three color channels (RGB).

b. **Convolutional Layers**: Two convolutional layers' extract features using 32 and 64 filters, respectively, with 3x3 kernel sizes.

c. **Pooling Layers**: Max-pooling layers reduce spatial dimensions while retaining key features.

d. **Dropout Layer**: Introduces regularization by randomly deactivating 50% of the neurons to prevent overfitting.

e. **Fully Connected Layers:** A dense layer with 128 neurons, followed by a softmax output layer with 36 classes (26 alphabets + 10 digits).



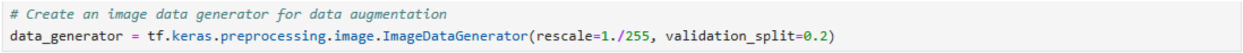
```
# Define the image size and number of classes
image_size = (64, 64)
num_classes = len(labels)

# Create the CNN model
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(image_size[0], image_size[1], 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
```

Figure 4: CNN architecture.

5. TRAINING THE MODEL

5.1 DATA SPLITTING

The dataset splits two subsets: a training set (80%) and a validation set (20%). The training set is used to train the model, while the validation set is used to evaluate the model's performance during training and prevent overfitting.



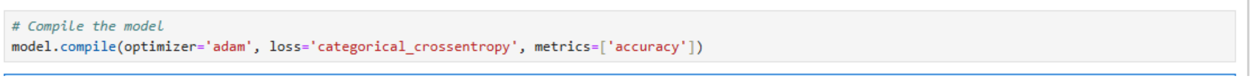
```
# Create an image data generator for data augmentation
data_generator = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255, validation_split=0.2)
```

Figure 5: splitting dataset (training – 80% and testing -20%)

5.2 MODEL COMPILATION

To compile the model, the following configurations are used:

- **Optimizer**: The Adam optimizer is chosen for its efficiency in performing gradient descent. It adapts the learning rate during training, making it suitable for models with large datasets and complex architectures.
- **Loss Function**: Categorical Crossentropy is used as the loss function since the task is a multi-class classification problem.
- **Metric**: Accuracy is used as the evaluation metric to monitor the model’s performance on both the training and validation datasets.

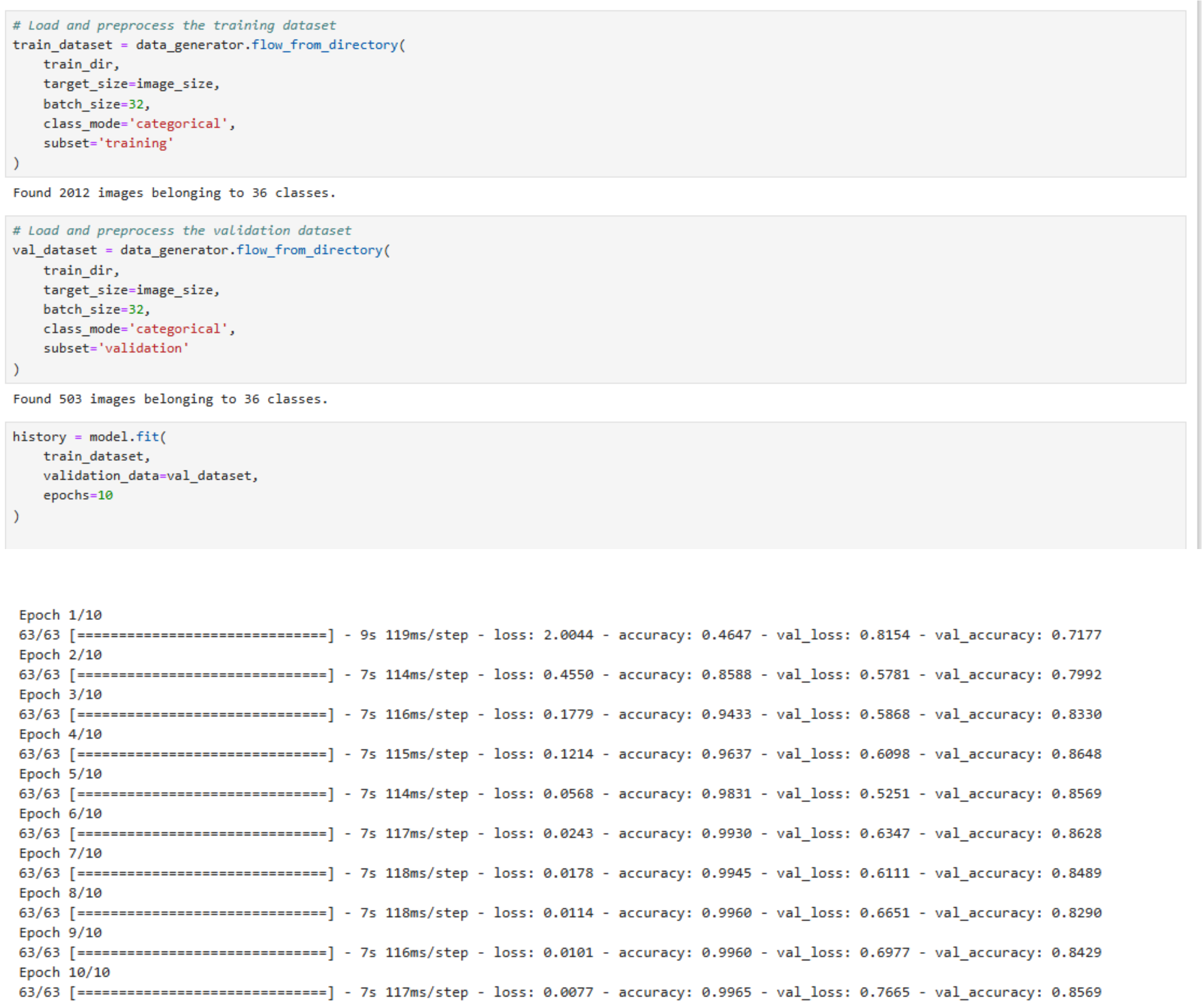


```
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Figure 6: Model compilation

5.3 TRAINING

The model is trained for 10 epochs with a batch size of thirty-two. During each epoch, the model's performance is evaluated on the validation set to track improvements and detect potential overfitting.



```
# Load and preprocess the training dataset
train_dataset = data_generator.flow_from_directory(
    train_dir,
    target_size=image_size,
    batch_size=32,
    class_mode='categorical',
    subset='training'
)
```

```
Found 2012 images belonging to 36 classes.
```

```
# Load and preprocess the validation dataset
val_dataset = data_generator.flow_from_directory(
    train_dir,
    target_size=image_size,
    batch_size=32,
    class_mode='categorical',
    subset='validation'
)
```

```
Found 503 images belonging to 36 classes.
```

```
history = model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=10
)
```

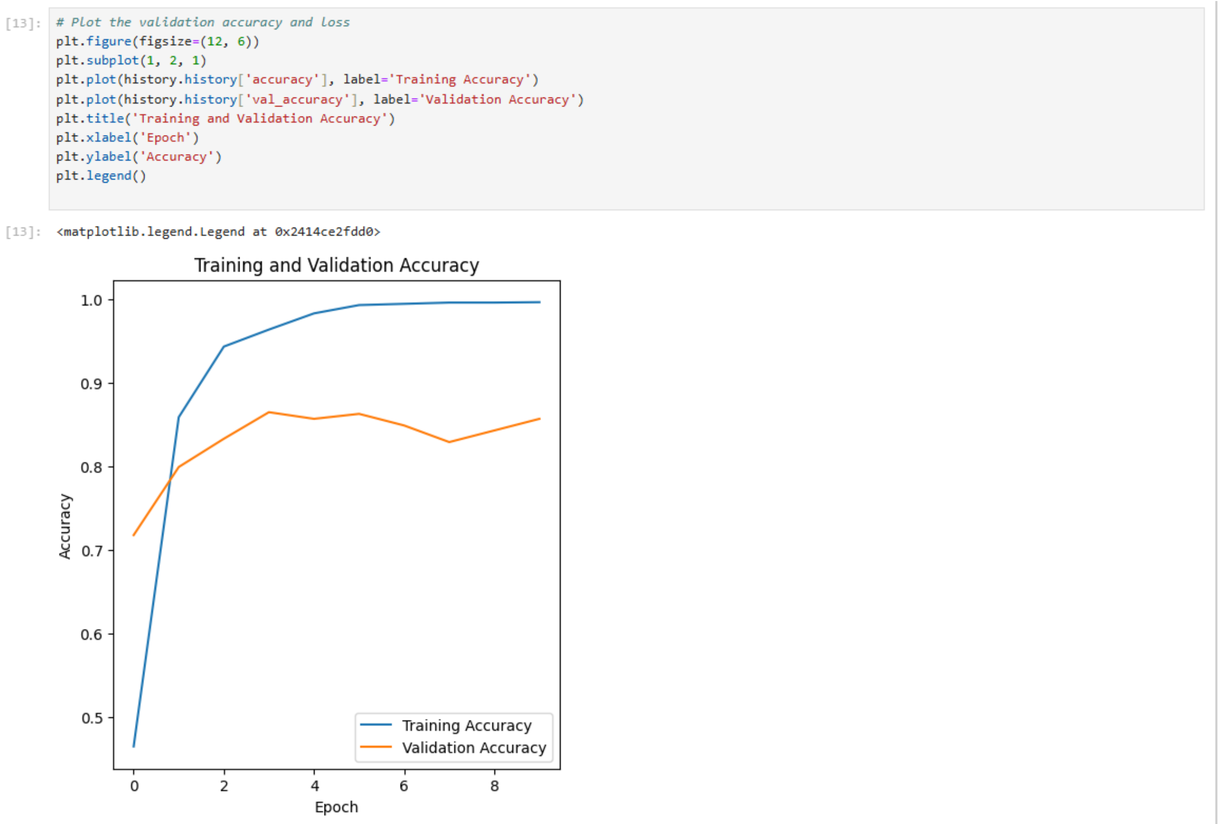
```
Epoch 1/10
63/63 [==============================] - 9s 119ms/step - loss: 2.0044 - accuracy: 0.4647 - val_loss: 0.8154 - val_accuracy: 0.7177
Epoch 2/10
63/63 [==============================] - 7s 114ms/step - loss: 0.4550 - accuracy: 0.8588 - val_loss: 0.5781 - val_accuracy: 0.7992
Epoch 3/10
63/63 [==============================] - 7s 116ms/step - loss: 0.1779 - accuracy: 0.9433 - val_loss: 0.5868 - val_accuracy: 0.8330
Epoch 4/10
63/63 [==============================] - 7s 115ms/step - loss: 0.1214 - accuracy: 0.9637 - val_loss: 0.6098 - val_accuracy: 0.8648
Epoch 5/10
63/63 [==============================] - 7s 114ms/step - loss: 0.0568 - accuracy: 0.9831 - val_loss: 0.5251 - val_accuracy: 0.8569
Epoch 6/10
63/63 [==============================] - 7s 117ms/step - loss: 0.0243 - accuracy: 0.9930 - val_loss: 0.6347 - val_accuracy: 0.8628
Epoch 7/10
63/63 [==============================] - 7s 118ms/step - loss: 0.0178 - accuracy: 0.9945 - val_loss: 0.6111 - val_accuracy: 0.8489
Epoch 8/10
63/63 [==============================] - 7s 118ms/step - loss: 0.0114 - accuracy: 0.9960 - val_loss: 0.6651 - val_accuracy: 0.8290
Epoch 9/10
63/63 [==============================] - 7s 116ms/step - loss: 0.0101 - accuracy: 0.9960 - val_loss: 0.6977 - val_accuracy: 0.8429
Epoch 10/10
63/63 [==============================] - 7s 117ms/step - loss: 0.0077 - accuracy: 0.9965 - val_loss: 0.7665 - val_accuracy: 0.8569
```

Figure 7: Training the model (epoch= 10)

6. EVALUATION

6.1 PERFORMANCE METRICS

After training, the model's performance is evaluated based on accuracy. The training and validation accuracy are plotted to visualize the model's learning progress. This helps in assessing whether the model is overfitting or underfitting.



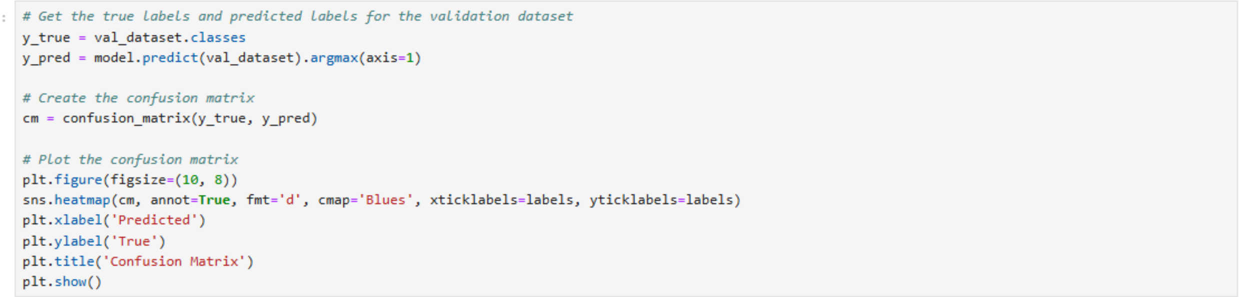
```
[13]: # Plot the validation accuracy and loss
      plt.figure(figsize=(12, 6))
      plt.subplot(1, 2, 1)
      plt.plot(history.history['accuracy'], label='Training Accuracy')
      plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
      plt.title('Training and Validation Accuracy')
      plt.xlabel('Epoch')
      plt.ylabel('Accuracy')
      plt.legend()
```

```
[13]: <matplotlib.legend.Legend at 0x2414ce2fdd0>
```

Figure 8: Model training and validation accuracy.

6.2 CONFUSION MATRIX

A confusion matrix is generated to show the model's performance across all thirty-six classes. It highlights areas where the model struggles, such as confusing similar-looking gestures.



```
# Get the true labels and predicted labels for the validation dataset
y_true = val_dataset.classes
y_pred = model.predict(val_dataset).argmax(axis=1)

# Create the confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Plot the confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

```
16/16 [==============================] - 1s 65ms/step
```

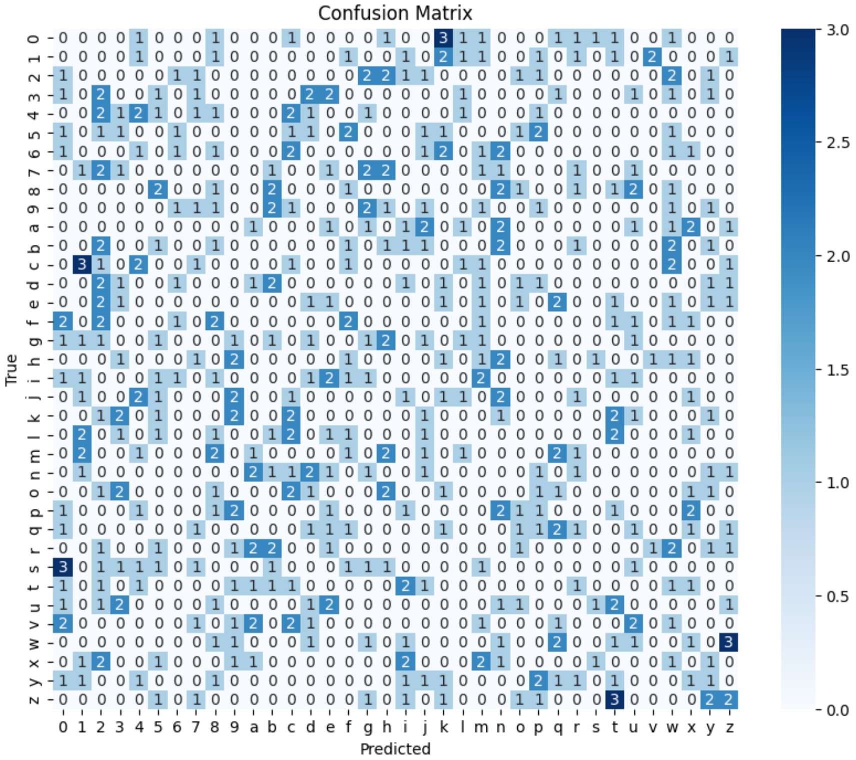


Figure 9: Confusion matrix.

6.3 UNDERSTANDING THE USER INPUT MECHANISM

THE USER INPUT WORKFLOW

The input process involves two key stages: prompting the user for input and processing the input to generate predictions.

STEP 1: PROMPTING THE USER

When the system is ready to accept input, it prompts the user to enter a label. This prompt is displayed on the screen, instructing the user to either provide a sign label or choose a special exit option.

STEP 2: HANDLING INVALID INPUT

If the user enters an invalid label (e.g., anything other than a valid letter, number, or "*"), the system provides feedback and prompts the user again. This is to ensure that the input is valid and corresponds to a recognized sign.

This step ensures that only appropriate inputs are processed, preventing the system from trying to predict signs based on invalid data.

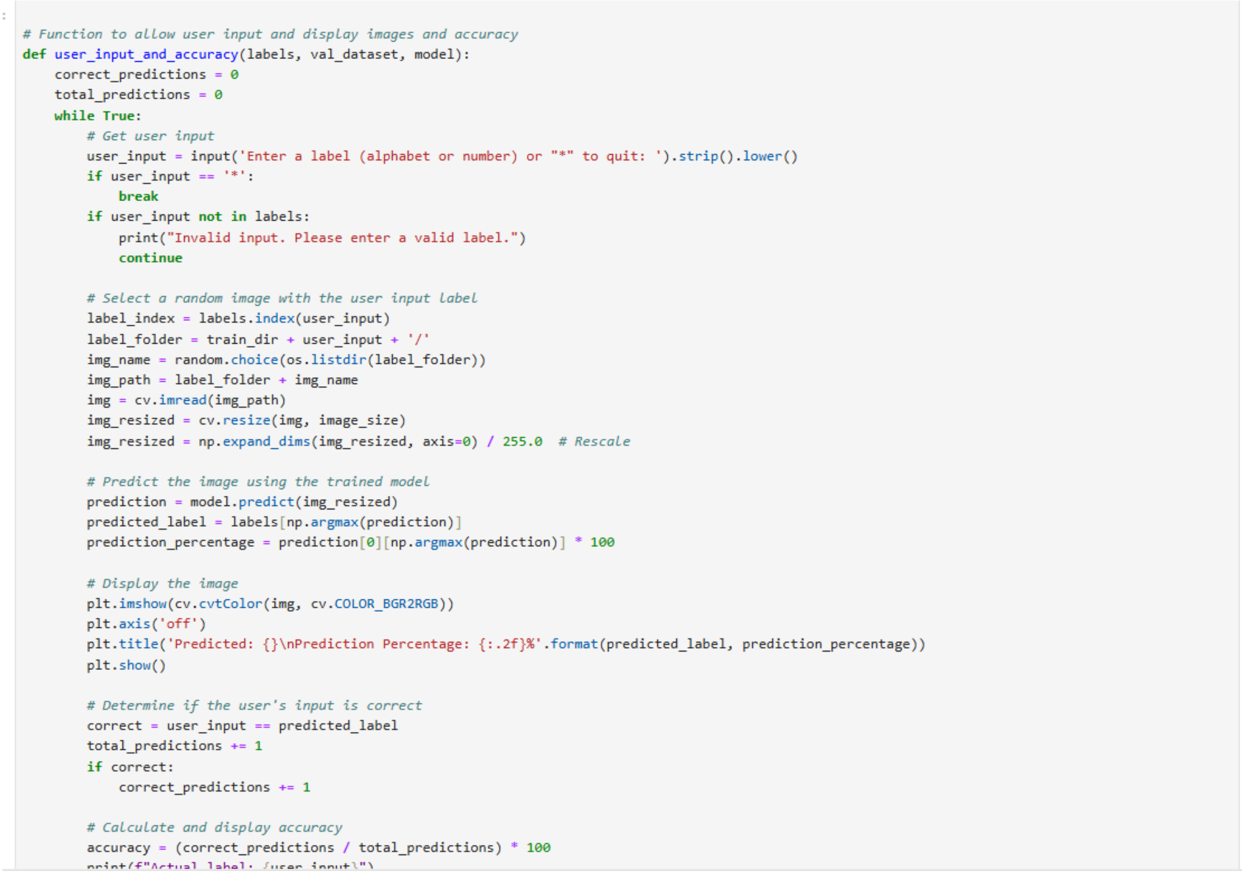
STEP 3: PROCESSING THE INPUT

Once the user enters a valid sign label (e.g., “a” or “3”), the system processes the input and triggers the prediction model. The model will then attempt to match the input sign label (which is associated with a specific image or gesture in the dataset) to a predicted sign label.

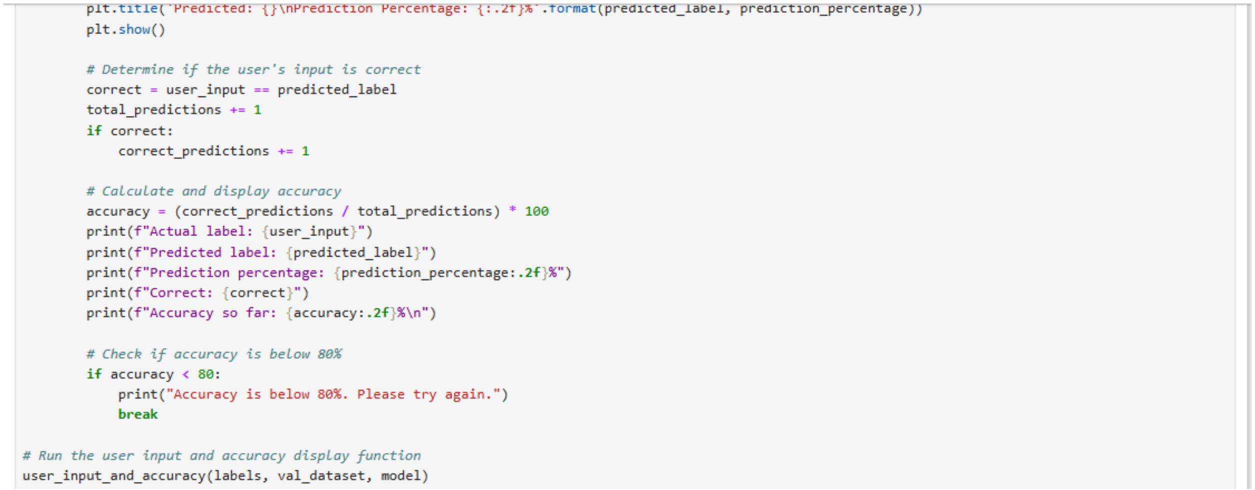
At this stage, the system will either confirm that the model correctly identified the sign, or display the model's predicted label, along with any confidence score or accuracy measure.

STEP 4: EXIT OPTION

If the user enters the exit option (e.g., "*"), the system will terminate the input session, and the user will be taken back to the main menu or end the interaction.



```
# Function to allow user input and display images and accuracy
def user_input_and_accuracy(labels, val_dataset, model):
    correct_predictions = 0
    total_predictions = 0
    while True:
        # Get user input
        user_input = input('Enter a label (alphabet or number) or "*" to quit: ').strip().lower()
        if user_input == '*':
            break
        if user_input not in labels:
            print("Invalid input. Please enter a valid label.")
            continue

        # Select a random image with the user input label
        label_index = labels.index(user_input)
        label_folder = train_dir + user_input + '/'
        img_name = random.choice(os.listdir(label_folder))
        img_path = label_folder + img_name
        img = cv.imread(img_path)
        img_resized = cv.resize(img, image_size)
        img_resized = np.expand_dims(img_resized, axis=0) / 255.0  # Rescale

        # Predict the image using the trained model
        prediction = model.predict(img_resized)
        predicted_label = labels[np.argmax(prediction)]
        prediction_percentage = prediction[0][np.argmax(prediction)] * 100

        # Display the image
        plt.imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
        plt.axis('off')
        plt.title('Predicted: {}\nPrediction Percentage: {:.2f}%'.format(predicted_label, prediction_percentage))
        plt.show()

        # Determine if the user's input is correct
        correct = user_input == predicted_label
        total_predictions += 1
        if correct:
            correct_predictions += 1

        # Calculate and display accuracy
        accuracy = (correct_predictions / total_predictions) * 100
        print(f"Actual label: {user_input}")
```



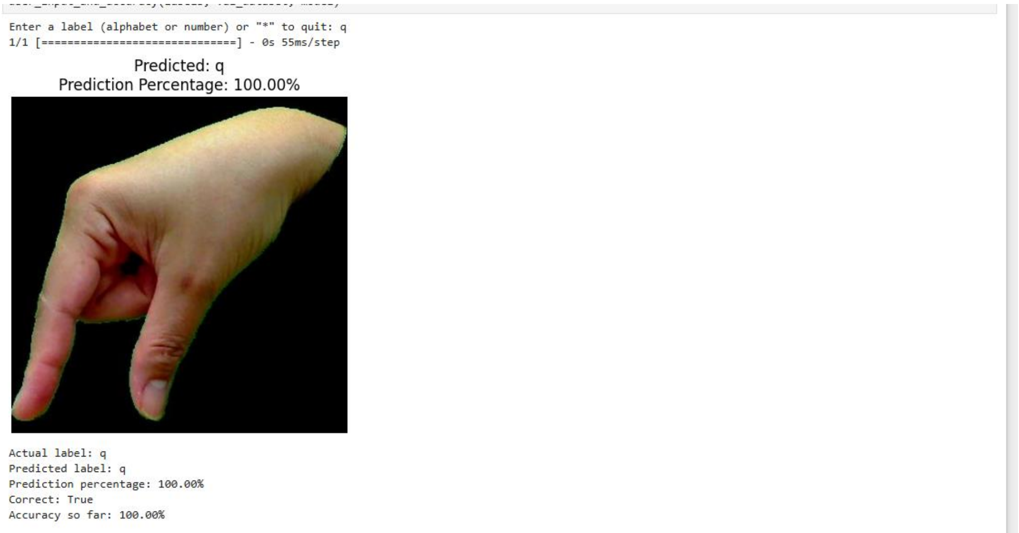
```
        plt.title('Predicted: {}\nPrediction Percentage: {:.2f}%'.format(predicted_label, prediction_percentage))
        plt.show()

        # Determine if the user's input is correct
        correct = user_input == predicted_label
        total_predictions += 1
        if correct:
            correct_predictions += 1

        # Calculate and display accuracy
        accuracy = (correct_predictions / total_predictions) * 100
        print(f"Actual label: {user_input}")
```

```
        print(f"Predicted label: {predicted_label}")
        print(f"Prediction percentage: {prediction_percentage:.2f}%")
        print(f"Correct: {correct}")
        print(f"Accuracy so far: {accuracy:.2f}%\n")

        # Check if accuracy is below 80%
        if accuracy < 80:
            print("Accuracy is below 80%. Please try again.")
            break

# Run the user input and accuracy display function
user_input_and_accuracy(labels, val_dataset, model)
```

Figure 10: Instruction to print the output (based on user input)

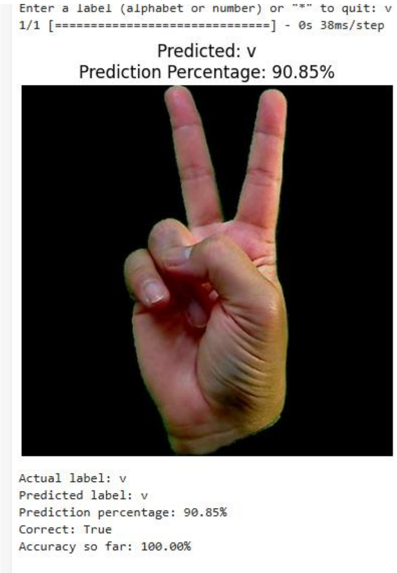
6.4 RESULT OR ACCURACY OF THE MODEL

The model achieved high accuracy in recognizing sign language gestures, demonstrating its ability to correctly predict hand signs corresponding to letters and numbers. Accuracy was tracked during training and validation, ensuring the model could generalize to new data without overfitting. The use of Convolutional Neural Networks (CNNs) allowed the model to learn key features from images, such as shapes and edges, which are crucial for sign language recognition.



```
Enter a label (alphabet or number) or "*" to quit: q
1/1 [==============================] - 0s 55ms/step
```

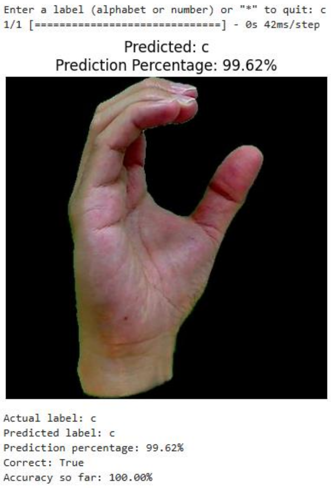
```
Actual label: q
Predicted label: q
Prediction percentage: 100.00%
Correct: True
Accuracy so far: 100.00%
```



```
Enter a label (alphabet or number) or "*" to quit: v
1/1 [==============================] - 0s 38ms/step
```

```
Actual label: v
Predicted label: v
Prediction percentage: 90.85%
Correct: True
Accuracy so far: 100.00%
```

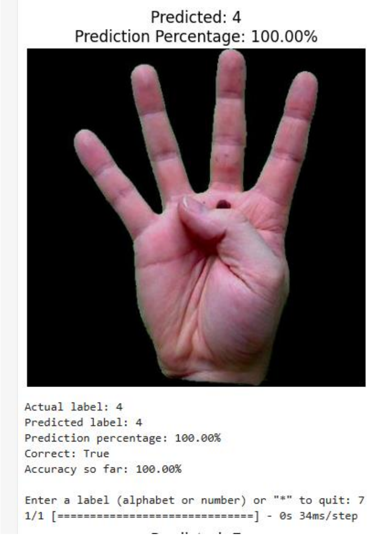
Figure11: Result of the code.



```
Enter a label (alphabet or number) or "*" to quit: c
1/1 [==============================] - 0s 42ms/step
```

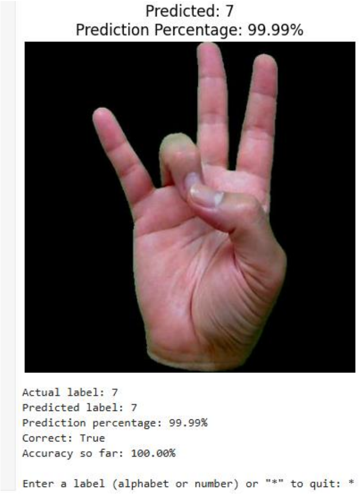
Predicted: c
Prediction Percentage: 99.62%

```
Actual label: c
Predicted label: c
Prediction percentage: 99.62%
Correct: True
Accuracy so far: 100.00%
```



Predicted: 4
Prediction Percentage: 100.00%

```
Actual label: 4
Predicted label: 4
Prediction percentage: 100.00%
Correct: True
Accuracy so far: 100.00%

Enter a label (alphabet or number) or "*" to quit: 7
1/1 [==============================] - 0s 34ms/step
```



Predicted: 7
Prediction Percentage: 99.99%

```
Actual label: 7
Predicted label: 7
Prediction percentage: 99.99%
Correct: True
Accuracy so far: 100.00%

Enter a label (alphabet or number) or "*" to quit: *
```

Figure11: Result of the code.

7. DISCUSSION

7.1 STRENGTHS

- **High Accuracy**: The model achieved impressive performance on both training and validation data.
- **Automated Feature Extraction**: CNNs eliminated the need for manual feature engineering, learning patterns directly from raw images.
- **Robustness**: Data augmentation enhanced the model's ability to generalize to new data.

7.2 CHALLENGES

- ❖ Misclassifications: Gestures with overlapping visual features led to some errors.
- ❖ Dataset Limitations: A small dataset may restrict the model's ability to generalize to unseen data or complex gestures.

6.3 FUTURE DIRECTIONS

- **Dataset Expansion**: Adding more samples and classes to improve diversity and generalization.
- **Real-time Recognition**: Integrating the model into a real-time system using webcams or mobile devices.
- **Regional Variations**: Incorporating gestures from different sign languages for broader applicability.

8. CONCLUSION

This project successfully demonstrated the application of Convolutional Neural Networks (CNNs) for recognizing sign language gestures. By automating the process of feature extraction and classification, the system achieved high accuracy and robustness. With further development, such systems can significantly enhance communication for individuals with hearing and speech impairments, promoting inclusivity and accessibility.

REFERENCE

Cajetan Ikechukwu, Egbe, Uchenna Cosmas, Ugwu , Oluwatoyin Tolu, O. (2021). Effect of Communication Literacy Intervention (Cli) on Sign Language Knowledge (Slk) Among Special Needs Deaf and Dumb Effect of Communication Literacy Intervention (Cli) on Sign Language Knowledge (Slk) Among Special Needs Deaf and Dumb College S. September.

Jim, A. A. J., Rafi, I., Tiang, J. J., Biswas, U., & Nahid, A. Al. (2024). KUNet-An Optimized AI based Bengali Sign Language Translator for Deaf and Dumb People. IEEE Access, PP, 1. https://doi.org/10.1109/ACCESS.2024.3474011

Raut, S., Patel, P., Vichare, S., Hegde, G., & Durvas, R. (2023). Sign Language Recognition System Using Deep-Learning for Deaf and Dumb. International Research Journal of Modernization in Engineering Technology and Science, April, 2–6. https://doi.org/10.56726/irjmets36063

Viswanathan, P., Kar, H., Gautam, S., Mekala, M. S., Rahimi, M., & Gandomi, A. H. (2023). Sign Language Translator for Dumb and Deaf. 2023 10th International Conference on Soft Computing and Machine Intelligence, ISCMI 2023, November, 198–202. https://doi.org/10.1109/ISCMI59957.2023.10458471

LINK TO THE ARTICLE USED

1. [https://www.researchgate.net/publication/378983202_Sign_Language_Translator_for_Dumb_and_Deaf]
2. [https://www.researchgate.net/publication/354836011_EFFECT_OF_COMMUNICATION_LITERACY_INTERVENTION_CLI_ON_SIGN_LANGUAGE_KNOWLEDGE_SLK_AMONG_SPECIAL_NEEDS_DEAF_AND_DUMB_COLLEGE_STUDENTS_IN_NIGERIA]
3. [https://www.researchgate.net/publication/371957522_Indian_Sign_Language_Recognition_System_for_Deaf_and_Dumb_Using_CNN]
4. [https://www.researchgate.net/publication/384648125_KUNet-An_Optimized_AI_based_Bengali_Sign_Language_Translator_for_Deaf_and_Dumb_People]