A Hand Gesture Recognition System Analysis

Janakan Sivaloganathan
Department of Electrical, Computer & Biomedical Engineering
Toronto Metropolitan University
Toronto, Canada
jsiva@torontomu.ca

Mithushan Sivarajah
Department of Electrical, Computer & Biomedical Engineering
Toronto Metropolitan University
Toronto, Canada
msivarajah@torontomu.ca

Dhaarshan Preshanthan
Department of Electrical, Computer & Biomedical Engineering
Toronto Metropolitan University
Toronto, Canada
dpreshanthan@torontomu.ca

***Abstract*—In the growing industry of human-computer interaction, the pursuit of more accessible forms of communication has always been a driving force. Hand Gesture Recognition Systems represent a groundbreaking stride in this direction. Unlike traditional interaction mechanisms which often rely on hardware mechanisms, hand gesture recognition offers a more seamless mode of interaction. In this report, we created a detection system that detects hand forms through preprocessed computer vision algorithms described in the methodology section of this report along with a Convolutional Neural Network (CNN) to classify these images. Our system can predict correct hand gestures at 56.28% accuracy. Our system can be found here: https://github.com/janakan2466/computer-vision-project**

***Index Terms*—hand gesture recognition, computer vision, convolutional neural network, human-computer interaction, image processing**

I. Introduction

The quest for more accessible forms of communication has led to the exploration of innovative technologies in human-computer interaction. In this context, hand gesture recognition systems have emerged as a groundbreaking solution, offering a more intuitive and natural mode of interaction. Such communicative technologies are increasingly embedded in a diverse set of commercial products which increases accessibility and reach. Although not a new topic, hand recognition products have been deployed globally and have become critical for many in their daily lives, so improvements in image detection are critical to reducing false negative reads.

In this paper, we investigate and employ several optimization techniques to improve the bottlenecks and scalability factors related to the accuracy of hand gesture recognition.

II. Past Work

Human-computer interaction is a domain of study to improve the interaction between users and computers through various sensory nodes such as gestures, speech, facial, and body expressions [4]. Hand gestures are a form of body language that can be displayed through the palm center, finger position, and shape constructed by the hand [2]. For vision-based systems, hand gestures can be classified into two types, static and dynamic. Static hand gestures are classified as gestures where the position does not change for an amount of time whereas dynamic gestures have movement changes within the given time [4]. A hand gesture recognition system is a computer vision system that leverages several algorithms to process image inputs from a camera capturing specified hand movements. conversions, to improve the quality of the image dataset and 2) feed the processed images into machine learning algorithms such as a Convolutional Neural Network (CNN) to learn the attributes from the images and leverage the learnings to classify different hand gestures.

The most common form of human gesture recognition systems is composed of two overall high-level steps: 1) process images to reduce the background noise, isolate the hand from the background image, and apply some colour space conversions, to improve the quality of the image dataset and 2) feed the processed images into machine learning algorithms such as a Convolutional Neural Network (CNN) to learn the attributes from the images and leverage the learnings to classify different hand gestures.

To increase the accuracy and efficacy of image segmentation and feature extraction methods, Victor Chang et al [3] proposed several denoising algorithms to filter the picture, decrease noise, and ensure that the hand is the most prominent figure in the image. Ranging from applying mean and median filtering to remove noise, to employing histogram of equalization to improve image quality, these techniques did not result in desired results for image enhancement. The image quality was improved by defocusing the background by finding histogram equalization optimization techniques after converting the red, green, and blue (RGB) multi-channel image to a hue, saturation, and value (HSV) colour space and retrieving the region of interest to extract a mask using the yen Otsu threshold. The recognition algorithm feeds the extracted hand area to a CNN that leverages a multi-layer architecture which is built upon an input layer, output layer, and hidden layer, which is composed of numerous convolutional layers, pooling layers, and fully linked layers [3].

Hand recognition systems are not just limited to standard filters and learning algorithms. For example, Pragati Garg et al. processed the 2D profile of the hand model using the Unscented Kalman filter (UKF) which minimizes the geometric error between the profiles and edges. UKF provided higher accuracy than other filtering algorithms such as the Kalman filter for high frame rate particle filtering. For the learning algorithm, Haar-like features to extract information from a region of interest were leveraged and then fed into the

AdaBoost scale-invariant learning algorithm to select the best features and combine them [9]. The learning model to classify the feature extractions can also vary as Paulo Trigueiros et al. devised a multi-class supervised learning model called the Support Vector Machine (SVM) which used the feature vectors derived from the feature extractions from the passed images of hand gestures for training. This also resulted in a successful image processing and classification of hand gestures [10].

Our work was built upon the 3-layer camera, detection, and interface hand gesture recognition system for human-computer interaction by Aashni Haria et al. [9]. The sequence of steps that was leveraged is highlighted in Table I.

TABLE I
HAND GESTURE RECOGNITION SYSTEM OF PROCESSES [2]

Steps	Processes
1	• Input is captured with a camera. • Image is converted to grayscale.
2	• Noise removal and smoothening of the image is done • Otsu and Inverted Binary thresholding are performed
3	• Contour Extraction is carried out • Convex Hull is found along with Convexity Defects
4	• Depending upon the convexity defects, gestures are recognized • For gestures without the exposure of finger
5	• Gesture-action pairs are mapped • If PowerPoint is open and the webcam detects a palm for 5 continuous frames, the dynamic gesture is detected

III. METHODOLOGY

Our hand gesture system is very similar to a previous work called "Hand Gesture Recognition for Computer Interaction" by Aashni Haria [4]. Instead of using a Haar Cascading Classifier, we used a Convolutional Neural Network (CNN) for our system. We also removed contour extraction and convex hull in our pre-processing algorithm. This is because this network is much more powerful and efficient compared to Haar Cascading. Keep your text and graphic files separate until after the text has been formatted and styled.

The following is the basic algorithm of our recognition system:

1) Import the dataset and resize all images
2) Convert images to grayscale
3) Apply Gaussian noise removal
4) Apply Otsu's thresholding and inverted binary thresholding
5) Apply Canny edge detection
6) Normalization of Pixels
7) Pair gestures using CNN

A. Import dataset and resize all images

This phase of our project entailed selecting and importing the dataset crucial for training our algorithm. The most challenging aspect of this stage was not the implementation, which was relatively straightforward, but rather the decision-making process regarding which dataset would best suit our requirements. Our primary considerations were the quality of the dataset and its alignment with the specific objectives of our algorithm. After a thorough evaluation, we selected the American Sign Language (ASL) dataset. This choice was driven by its comprehensive coverage and high relevance to our goals.

Additionally, to optimize the dataset for our algorithm, we standardized the size of each image to 320x320 pixels. This uniformity ensures consistency in input data, aiding in more effective training and evaluation of the model. An example from the ASL dataset, shown in Figure 1 below, illustrates the type of images we are working with. This step is pivotal as it lays the foundational groundwork for the subsequent stages of our algorithm's development.

Fig. 1. Original image in dataset.

B. Convert Images to Grayscale

Converting images to grayscale allows us to use a simplified algorithm, it also reduces the computational requirements that are required [5]. It reduces the overall complexity of the image whilst also simplifying the problem that the algorithm is responsible for solving. It is possible to complete this project without this step, but this step makes it much easier to handle and makes the algorithm more efficient. Figure 2 is an example of converting an image to a grayscale. This step makes use of the following function: `gray = cv2.cvtColor(resized\_image,cv2.COLOR\_BGR2GRAY)`. The code above uses the following formula [11]:

$$Gray = 0.299Red + 0.587Green + 0.114Blue \quad (1)$$

C. Gaussian Noise Removal

Gaussian blur plays a crucial role in image processing by addressing the intricate high-frequency elements present within an image that might be misinterpreted as edges [6]. Figure 3 is an example of Gaussian Noise removal being applied. This process is essential as it helps to reduce the

Fig. 2. Grayscale image.

impact of these components on the final image output. The utilization of a low pass filter is aimed at smoothing and denoising the image while striving to maintain the authenticity and fidelity of the original visual data intact. This particular stage serves as a catalyst, significantly enhancing the effectiveness of subsequent image processing steps, and enabling them to yield more accurate and refined results. This step makes use of the following function: `blurred = cv2.GaussianBlur(gray, (5, 5), 0)`. The code above uses the following formula [12]:

$$G(x, y) = (1/(2 * \pi * \sigma)) * e - (x2 + y2)/2\sigma 2 \quad (2)$$

Fig. 3. Grayscale image.

D. Otsu's and Inverted Binary Thresholding

This technique allows the algorithm to attempt to separate the focal object from the background in the image [2]. Figure 4 is an example of Otsu's thresholding and inverted binary thresholding being applied to an image. It develops a binary image in which each pixel is classified as foreground (object of interest) or background. This step is necessary to isolate the hand in our image and enhance the relevant areas of the image. This meticulous process significantly refines the subsequent enhancement techniques applied to the isolated object, ensuring a more detailed and accurate result. This step makes use of the following function: `thresholded=cv2.threshold(blurred,0,255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)`.

Fig. 4. Threshold image.

E. Canny Edge Detection

Canny Edge Detection is the edge detection method we chose to use in our algorithm, it outclasses Sobel and Prewitt edge detection methods [7]. It accurately identifies any edges present in the image. This method also includes a step in which it connects separately detected edge segments together to form a continual segment, this improves the overall completeness of the result. The precision and adaptability of Canny Edge Detection significantly contribute to its superiority over other conventional edge detection techniques. Figure 5 demonstrates Canny Edge Detection being applied to an image. This step makes use of the following function: `edges = cv2.Canny(thresholded, 50, 150)`. The process this function undergoes is as follows:

1) Finding the Intensity Gradient of the Image: Filter the image with a Sobel kernel along the x and y axis to get the first derivative in each direction. Find the gradient and direction of each pixel using the two images Gx and Gy by implementing the following formulas [8]:

$$\begin{aligned} Edge\,Gradient(G) &= \sqrt{{G^2}_{\mathrm{x}} + {G^2}_{\mathrm{y}}} \\ Angle(\theta) &= \tan^{-1}\frac{G_{\mathrm{y}}}{G_{\mathrm{x}}} \end{aligned} \quad (3)$$

2) Non-maximum Suppression: Scan images to find and remove undesirable pixels that are not needed for edges. It does a check to determine if the pixel is a local maximum along the gradient's direction. This step results in a binary image of thin edges [8].

3) Hysteresis Thresholding: This step is responsible for determining which of the edges in the previously generated image are real edges. By assigning a minimum and maximum threshold value, this step keeps edges that are above a maximum threshold value and abandons edges that are lower than the minimum threshold value. It then determines what edges between the minimum and maximum threshold value are real edges based on their relationships with other real edges. If they are connected to real edge pixels they are also considered as a real edge. This step also removes small noise since it assumes that edges are long lines [8].

Fig. 5. Canny edge detector.

F. Normalization of Pixels

Normalization is a preprocessing technique used to standardize the range of independent variables of features of data. This typically involves scaling pixel values to the range [0,1]. This is beneficial because neural networks learn faster with smaller, normalized inputs. Having a consistent scale for all input features means the learning rate will affect all features equally.

G. Pair Gestures Using CNN

A Convolutional Neural Network is an algorithm that can take an input image, learn features about the image, and be able to differentiate one from the other [11]. Figure 6 illustrates an example of a CNN.

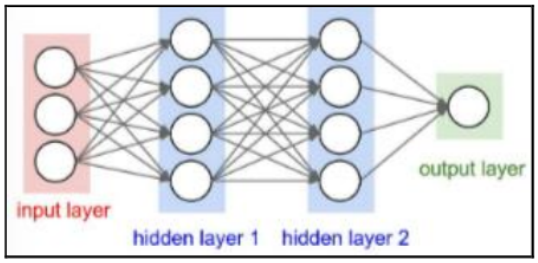

Fig. 6. Example of a Convolutional Neural Network.

The input layer corresponds to the pixels of the input image. For our grayscale images, we will have one input neuron for each pixel. Since we have 320x320 pixel resolution, we will have 102,400 neurons. The hidden layers are where feature extraction occurs. As the data passes through these layers, the network begins to recognize features in the images such as edges. The output layer corresponds to the number of classes. In our detection, we will have 26 classes, one for each letter in the alphabet.

The following is the architecture of the CNN model which can be found in Appendix B:

1) *Sequential Model*: The model uses a Keras' Sequential model. It's a model that is composed of a stack of layers.
2) *Convolutional Layers*: These layers will process the input images. The first layer will use 32 filters with a kernel size of 5x5. The next layer will use 64 filters with a kernel size of 3x3. Both layers are responsible for extracting features from the images.
3) *Pooling Layers*: These layers are used to reduce the spatial dimensions of the output volume. It helps to reduce computational load and controls overfitting.
4) *Flatten Layer*: This layer flattens the output from the convolutional and pooling layers so that it can be used as input to the fully connected layers. Dense Layers: These layers connect every neuron to every neuron in the previous layer.
5) *Output Layer*: Provides a probability distribution over the 26 classes.

IV. Approach

We collaborated on a hand gesture recognition algorithm using Jupyter Notebooks on Google Collab. We decided to use Python3 and OpenCV to perform image processing. The source code for all steps and additional documentation is provided in Appendix Section B and we used Filipe Borba's hand recognition model to help us create our own CNN model [1].

V. Experimental Results

The accuracy of our CNN model is a modest 56.28%. Figure 7 shows the predictions of each letter in the alphabet and focuses on individual predictions and their correctness. The blue label indicates that it was predicted correctly, and the red label indicates the prediction was incorrect. The "True" label indicates the class of the image, or in other words, the correct answer. The "Predicted" label indicates the model's prediction of an image and the percentage along with it indicates the model's confidence in its prediction. For example, in red it states "Predicted: M 95% (True: Y)". This means the model incorrectly predicted the image of the letter 'Y' as the letter 'M' with a high confidence of 95%.

Our model's relatively low accuracy can be attributed to several factors, with background noise being a primary concern. As illustrated in Figure 1, the images do not have a uniform white background, which introduces unwanted variability. This issue becomes more evident in Figures 4 and 5, where both Otsu's Thresholding and the Canny edge detector inadvertently pick up background elements. These extraneous features in the background likely interfere with the model's ability to accurately discern and classify the intended subjects in the images.

Appendix Section A illustrates the confusion matrix of this model. A confusion matrix is a table used to describe the performance of a classification model. Each element of the matrix represents the count of predictions made by the model compared to the actual labels. Each row of the matrix corresponds to an 'Actual' class label. For example, the row labeled "Actual A" represents all the samples that are truly in class 'A' in the dataset ('y_test'). The Columns represent "Predicted" Labels. Each column represents a 'Predicted' class label. For instance, the column labeled "Predicted A" shows the predictions made by the model as the letter "A" regardless

Fig. 7. CNN prediction of each label.

of their actual letter. The values in each box can be categorized as Diagonal Elements or Off-Diagonal Elements. The Diagonal Elements are diagonal values ranging from the top-left of the matrix to the bottom-right of the matrix. Each value represents the number of times the model correctly predicted each letter. For example, a value 16 at the intersection of 'Actual A' and 'Predicted A' denotes that the model accurately identified the letter 'A' 16 times. The Off-Diagonal elements show instances where the model's predictions did not match the actual labels. For instance, value 3 in the box at the intersection of "Actual A" and "Predicted B" indicates how many times the model incorrectly predicted an 'A' as a 'B'.

Figure 8 shows the summary of a statistical summary of all key metrics like precision, recall, and F1- score for each class, as well as overall averages. This gives a quantitative evaluation of the model's performance. The columns can be split into 4:

1) *Precision*: Ratio of correctly predicted positive observations to the total predicted positives for each class.
2) *Recall*: Ratio of correctly predicted positive observations to all observations in actual class.
3) *F1-Score*: Weighted average of Precision and Recall, taking both false positives and false negatives into account.
4) *Support*: Number of actual occurrences of the class in the specified dataset.

The precision and recall values are the lowest in Class 'Y' and highest in Class F'. This suggests that 'F' was the easiest sign for the model to detect while 'Y' was the hardest to detect. This could be because of various factors including the quality of data, the complexity of sign, or the model's architecture.

Compared to Haar Cascading Classifier our model has a lower accuracy percentage. However, our model can do more complex gestures and can compute a larger dataset than their model.

Classification Report:

	precision	recall	f1-score	support
A	0.58	0.63	0.60	30
B	0.59	0.53	0.56	30
C	0.59	0.67	0.62	30
D	0.64	0.47	0.54	30
E	0.52	0.50	0.51	30
F	0.74	0.83	0.78	30
G	0.75	0.60	0.67	30
H	0.71	0.67	0.69	30
I	0.60	0.60	0.60	30
J	0.66	0.70	0.68	30
K	0.57	0.67	0.62	30
L	0.57	0.80	0.67	30
M	0.65	0.57	0.61	30
N	0.52	0.53	0.52	30
O	0.50	0.40	0.44	30
P	0.57	0.57	0.57	30
Q	0.55	0.57	0.56	30
R	0.53	0.57	0.55	30
S	0.57	0.53	0.55	30
T	0.50	0.40	0.44	30
U	0.47	0.47	0.47	30
V	0.42	0.43	0.43	30
W	0.45	0.57	0.50	30
X	0.35	0.37	0.36	30
Y	0.46	0.37	0.41	30
Z	0.63	0.63	0.63	30
accuracy			0.56	780
macro avg	0.56	0.56	0.56	780
weighted avg	0.56	0.56	0.56	780

Fig. 8. Classification report for the CNN model.

Overall, our results show the practicality of our algorithm against various hand gestures. Despite the algorithm only having a 56.28% accuracy, the model does very well on most tests as seen in Figure 7. The model will have drastic improvements with a simpler dataset and after accounting for background noise.

VI. Conclusion

The primary objective of this assignment was to develop a hand gesture recognition system capable of detecting American Sign Language. This involved employing a range of pre-processing techniques and utilizing a Convolutional Neural Network for gesture classification.

We achieved a 56.28% accuracy rate in correctly identifying hand gestures. While this may appear modest, it marks a significant milestone in our initial experience in machine learning. The system encountered challenges in letter detection, likely influenced by background noise interfering with edge detection. Future enhancements will focus on implementing pre-processing algorithms to negate background noise and accommodate a wider range of skin tones.

A pivotal challenge was enhancing the system's accuracy beyond 26%. This initial hurdle was largely due to the incorporation of contour extraction and convex hull techniques,

leading to a common machine learning pitfall known as 'overfitting.' This occurs when a model, overly complex relative to the dataset, memorizes rather than generalizes the data.

There are substantial opportunities for improving this recognition system. A key enhancement would be transitioning from a CNN to a YOLO (You Only Look Once) object detection model. YOLO's efficiency and capability for real-time object detection align perfectly with today's world.

Ultimately, we aim to create a hand gesture recognition system to detect various gestures. This helps us gain a deeper understanding of computer vision and machine learning. Our project provides satisfactory results and has helped us achieve all our objectives.

REFERENCES

[1] Borba, F. (2019, May 6). Hand gesture recognition using Machine Learning. GitHub.

[2] Cao, Q., Qingge, L., & Yang, P. (2021, October 21). Performance Analysis of Otsu-based Thresholding Algorithms: A comparative study. *Journal of Sensors*.

[3] Chang, V., Eniola, R. O., Golightly, L., & Xu, Q. A. (2023a, June 12). An exploration into human–computer interaction: Hand gesture recognition management in a challenging environment - SN computer science. *SpringerLink*.

[4] Haria, A., & Subramanian, A. (2017, October 16). Hand gesture recognition for Human Computer Interaction. *Hand Gesture Recognition for Human Computer Interaction*.

[5] Kanan, C., & Cottrell, G. W. (2012). Color-to-grayscale: Does the method matter in image recognition? *PloS one*.

[6] Lahiri, R. (2020, July 28). Gaussian blurring and its importance in image processing. *Medium*.

[7] M, V. (2022, August 9). Comprehensive guide to edge detection algorithms. *Analytics Vidhya*.

[8] OpenCV. (2023, December 13). Canny edge detection.

[9] P. Garg, N. Aggarwal, and S. Sofat, "Vision Based Hand Gesture Recognition," *Vision Based Hand Gesture Recognition*.

[10] P. Trigueiros, F. Ribeiro, and L. P. Reis, "Hand Gesture Recognition System Based in Computer Vision and Machine Learning," *Hand Gesture Recognition System based in Computer Vision and Machine Learning*.

[11] Rosebrock, A. (2023, June 8). OpenCV color spaces (cv2.cvtcolor). *PyImageSearch*.

[12] Sharda, A. (2021, January 25). Image filters: Gaussian blur. *Medium*.

[13] Saha, S. (2018, December 15). A comprehensive guide to Convolutional Neural Networks - the eli5 way. *Medium*.

APPENDIX

A. Confusion Matrix of the Convolutional Neural Network Model

	Predicted A	Predicted B	Predicted C	Predicted D	Predicted E	Predicted F	Predicted G	Predicted H	Predicted I	Predicted J	Predicted K	Predicted L	Predicted M	Predicted N
Actual A	19	4	1	1	1	0	0	0	0	0	0	1	0	1
Actual B	1	16	0	3	5	0	0	0	1	0	0	1	0	0
Actual C	0	0	20	3	2	0	2	0	1	0	1	0	0	0
Actual D	1	1	5	14	4	2	0	0	0	0	0	1	0	0
Actual E	7	3	0	0	15	1	0	0	1	0	1	0	0	0
Actual F	1	1	0	1	0	25	0	0	1	0	0	1	0	0
Actual G	0	1	0	0	0	0	18	6	1	3	0	0	0	0
Actual H	0	0	1	0	0	0	2	20	1	1	0	0	0	1
Actual I	1	0	0	0	0	0	0	0	18	2	3	3	0	0
Actual J	0	0	0	0	0	1	0	2	0	21	1	1	1	0
Actual K	0	1	0	0	0	0	0	0	1	4	20	1	1	0
Actual L	0	0	0	0	0	0	0	0	1	0	3	24	0	0
Actual M	1	0	0	0	0	0	0	0	0	0	0	3	17	5
Actual N	2	0	1	0	0	1	1	0	0	0	0	0	3	16
Actual O	0	0	0	0	0	1	0	0	0	0	0	2	1	3
Actual P	0	0	1	0	0	0	1	0	0	0	0	0	0	1
Actual Q	0	0	0	0	0	0	0	0	0	0	0	2	0	1
Actual R	0	0	1	0	0	1	0	0	0	0	3	0	1	0
Actual S	0	0	0	0	0	0	0	0	1	0	1	0	0	2
Actual T	0	0	1	0	0	2	0	0	1	0	0	0	0	0
Actual U	0	0	0	0	0	0	0	0	0	0	1	1	1	0
Actual V	0	0	1	0	0	0	0	0	0	0	1	0	0	0
Actual W	0	0	0	0	2	0	0	0	0	0	0	0	0	1
Actual X	0	0	0	0	0	0	0	0	0	0	0	0	1	0
Actual Y	0	0	0	0	0	0	0	0	2	1	0	0	0	0
Actual Z	0	0	2	0	0	0	0	0	0	0	0	1	0	0

Predicted O	Predicted P	Predicted Q	Predicted R	Predicted S	Predicted T	Predicted U	Predicted V	Predicted W	Predicted X	Predicted Y	Predicted Z
0	0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	0	1	1	1	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0
0	2	0	0	1	0	0	0	0	0	1	0
0	0	0	1	1	0	0	1	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	0	0	1	0	1	0
0	0	2	0	0	0	0	1	1	0	1	1
12	0	0	0	1	1	1	1	1	3	2	1
0	17	5	1	2	0	1	0	0	1	0	0
2	8	17	0	0	0	0	0	0	0	0	0
0	1	1	17	1	0	3	0	0	1	0	0
0	0	0	1	16	2	2	0	1	3	0	1
0	0	4	0	2	12	1	0	1	2	1	3
1	0	1	2	0	2	14	3	3	1	0	0
0	0	0	4	0	0	2	13	5	0	3	1
1	0	0	1	0	0	2	4	17	2	0	0
1	1	0	3	2	2	1	4	1	11	2	1
0	0	0	2	1	2	1	2	2	4	11	2
1	0	0	0	0	1	0	0	2	2	2	19

Fig. 9. Confusion matrices: A to N (top) and O to Z (bottom).

B. Code

```
#!/usr/bin/env python
# coding: utf-8
# cv_algorithm.py

# Import necessary libraries
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from collections import defaultdict
from google.colab import drive
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.callbacks import LearningRateScheduler
from sklearn.metrics import classification_report, confusion_matrix
from tabulate import tabulate

# Install kaggle and upload JSON file
!pip install -q kaggle
from google.colab import files
files.upload()

# Unzip dataset
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
!kaggle datasets download -d grassknoted/asl-alphabet
!unzip asl-alphabet.zip

# Mount Google Drive
drive.mount("/content/drive")

# Define constants
MAX_IMAGES_PER_LETTER = 100
DESIRED_SIZE = (320, 320)
TEST_SIZE = 0.3

# Set up a dictionary to count images per letter
image_count_per_letter = defaultdict(int)

# Set the start directory
start_directory = "./asl_alphabet_train"

# List to store the names of subdirectories
subdirectories = []

# Subdirectories to skip
skip_dirs = {"del", "nothing", "space"}

# Function to process and plot the final image
def process_and_plot_final_image(path):
    # Load the image
    image = cv2.imread(path)
    if image is None:
        raise ValueError("Image not found or path is incorrect")

    # Resize the image
    resized_image = cv2.resize(image, DESIRED_SIZE)

    # Convert to grayscale
    gray = cv2.cvtColor(resized_image, cv2.COLOR_BGR2GRAY)

    # Apply Gaussian blur
```

```
    blurred = cv2.GaussianBlur(gray, (3, 3), 0)

    # Apply Otsu's thresholding
    _, thresholded = cv2.threshold(blurred, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

    # Canny edge detection
    final_image = cv2.Canny(thresholded, 50, 150)

    # Normalize the final image
    normalized_image = final_image / 255.0

    # Display the final image
    plt.imshow(normalized_image, cmap="gray")
    plt.title("Final Processed Image")
    plt.xticks([]), plt.yticks([]) # Hide tick marks
    plt.show()

# Function to validate and plot images
def validate_26_images(predictions_array, true_label_array, img_array):
    # Array for pretty printing and figure size
    class_names = ["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"]
    plt.figure(figsize=(15, 15))

    for i in range(1, 27):
        # Assign variables
        prediction = predictions_array[i]
        true_label = true_label_array[i]
        img = img_array[i]

        # Plot in a good way
        plt.subplot(7, 4, i)
        plt.grid(False)
        plt.xticks([])
        plt.yticks([])
        plt.imshow(img, cmap=plt.cm.binary)

        predicted_label = np.argmax(prediction)

        # Change color of title based on good prediction or not
        color = "blue" if predicted_label == true_label else "red"

        plt.xlabel("Predicted: {} {:2.0f}% (True: {})".format(
            class_names[predicted_label], 100 * np.max(prediction), class_names[true_label]),
            color=color
        )
    plt.show()

# Load image paths and preprocess images
imagepaths = []
for root, dirs, files in os.walk(start_directory, topdown=False):
    for dir in dirs:
        if dir in skip_dirs:
            continue
        subdirectory = os.path.join(root, dir)
        subdirectories.append(subdirectory)
    for name in files:
        if name.lower().endswith(".jpg"):
            letter = os.path.basename(root)
            if letter in skip_dirs:
                continue
            if image_count_per_letter[letter] < MAX_IMAGES_PER_LETTER:
                path = os.path.join(root, name)
                imagepaths.append(path)
                image_count_per_letter[letter] += 1
                if image_count_per_letter[letter] == MAX_IMAGES_PER_LETTER:
                    continue
```

```
# Print information about collected image paths
print("Number of image paths:", len(imagepaths))
print(f"Collected image paths, with up to {MAX_IMAGES_PER_LETTER} images per letter.")

# Print subdirectories
print("Subdirectories found:")
for subdir in subdirectories:
    print(subdir)

# Process and plot a sample image
process_and_plot_final_image(imagepaths[2])

# Mount Google Drive
drive.mount("/content/drive")

# Apply computer vision algorithms to all pictures in the dataset
X = [] # Image data
y = [] # Labels
skipped_images = 0

# Loop through image paths to load images and labels into arrays
for path in imagepaths:
    # Load the image
    image = cv2.imread(path)
    if image is None:
        raise ValueError("Image not found or path is incorrect")

    # Resize the image to the desired size
    resized_image = cv2.resize(image, DESIRED_SIZE)

    # Convert to grayscale
    gray = cv2.cvtColor(resized_image, cv2.COLOR_BGR2GRAY)

    # Apply Gaussian blur
    blurred = cv2.GaussianBlur(gray, (3, 3), 0)

    # Apply Otsu's thresholding
    _, thresholded = cv2.threshold(blurred, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

    # Canny edge detection
    final_image = cv2.Canny(thresholded, 50, 150)

    # Add the processed image to the dataset
    X.append(final_image)

    # Process the label in the image path
    filename = os.path.basename(path)
    label_part = filename.split(".")[0]

    label_mapping = {
        "A": 0, "B": 1, "C": 2, "D": 3, "E": 4, "F": 5, "G": 6, "H": 7,
        "I": 8, "J": 9, "K": 10, "L": 11, "M": 12, "N": 13, "O": 14,
        "P": 15, "Q": 16, "R": 17, "S": 18, "T": 19, "U": 20, "V": 21,
        "W": 22, "X": 23, "Y": 24, "Z": 25,
    }

    if len(label_part) > 1 and label_part[1:].isdigit():
        label_char = label_part[0]
        label = label_mapping.get(label_char, -1)
    else:
        label = label_mapping.get(label_part, -1)

    y.append(label)

# Convert X and y into np.array to speed up future processing
X = np.array(X, dtype="uint8")
```

```
X = X / 255.0 # Normalize pixel values to be between 0 and 1
X = X.reshape(len(imagepaths) - skipped_images, DESIRED_SIZE[0], DESIRED_SIZE[1], 1) # Adjust
    for 1 channel

y = np.array(y)

print("Images loaded: ", len(X))
print("Labels loaded: ", len(y))

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=TEST_SIZE,
    random_state=42, stratify=y)

# Define a learning rate schedule
def lr_schedule(epoch):
   lr = 0.001
   if epoch > 30:
      lr *= 0.5
   if epoch > 60:
      lr *= 0.5
   return lr

# Create a LearningRateScheduler callback
lr_scheduler = LearningRateScheduler(lr_schedule)

# Create and compile the model with corrected input shape
model = Sequential([
   Conv2D(32, (5, 5), activation="relu", input_shape=(320, 320, 1)),
   MaxPooling2D((2, 2)),
   Conv2D(64, (3, 3), activation="relu"),
   MaxPooling2D((2, 2)),
   Conv2D(64, (3, 3), activation="relu"),
   MaxPooling2D((2, 2)),
   Flatten(),
   Dense(128, activation="relu"),
   Dense(26, activation="softmax"),
])

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])

# Filter out samples with '-1' labels
valid_train_samples = y_train != -1
valid_test_samples = y_test != -1

X_train_filtered = X_train[valid_train_samples]
y_train_filtered = y_train[valid_train_samples]

X_test_filtered = X_test[valid_test_samples]
y_test_filtered = y_test[valid_test_samples]

# Train the model with the filtered datasets and the learning rate scheduler
model.fit(
   X_train_filtered,
   y_train_filtered,
   epochs=20,
   batch_size=64,
   verbose=2,
   validation_data=(X_test_filtered, y_test_filtered),
   callbacks=[lr_scheduler],
)

# Save the entire model to an HDF5 file
model.save("project_ASL_recognition.keras")

# Make predictions on the test set
y_pred = model.predict(X_test)
```

```
# Convert predictions from probabilities to class labels
y_pred_classes = np.argmax(y_pred, axis=1)

# Generate a classification report
report = classification_report(y_test, y_pred_classes, target_names=class_names)

print("Classification Report:\n", report)

# Plot validation images
validate_26_images(y_pred, y_test, X_test)

# Assuming y_test contains the true labels and y_pred contains the predicted labels
confusion = confusion_matrix(y_test, y_pred)

# Create a DataFrame for the confusion matrix with labeled columns and indices
confusion_df = pd.DataFrame(
   confusion,
   columns=[f"Predicted {class_name}" for class_name in class_names],
   index=[f"Actual {class_name}" for class_name in class_names],
)

# Display the confusion matrix as a formatted table
print(tabulate(confusion_df, headers="keys", tablefmt="fancy_grid"))
```