## Introduction:

The aim of this project is to develop and evaluate methods for recognizing hand gestures using both traditional computer vision techniques and Convolutional Neural Networks (CNNs) and compare their performance. The following assignment focuses on understanding how both approaches perform and their limitations by handling the classification of 10 distinct hand gestures.

Classical computer vision techniques are based on the use of manually defined features and specific algorithms to extract meaningful patterns from images. These methods simplify the image by focusing on important geometric features, which are then analyzed to classify objects or patterns. Techniques such as edge detection, thresholding, and contour analysis are commonly used to transform raw image data into simplified representations that highlight these features. Although these methods can work effectively in controlled environments or with carefully prepared datasets, they often struggle to generalize when applied to more complex, diverse, or noisy real-world images. This is because classical techniques rely heavily on predefined rules and logic, which limits their adaptability in handling variations in the data.

CNNs can capture complex patterns like shapes, textures, and spatial relationships because they are built to automatically learn hierarchical features straight from raw image data. Unlike classical methods, which rely on manually defined features, CNNs use layers of convolutional filters that progressively extract more complex patterns. While deeper layers in the network capture more abstract and spatially meaningful representations like shapes and hierarchical structures, early layers recognise fundamental features like edges and textures. This ability to generalize across diverse datasets has made CNNs the state-of-the-art approach for tasks like object detection, classification, and segmentation. However, this advanced capability comes at the cost of requiring significant computational power and large datasets, setting them apart from the simpler, less resource-intensive nature of classical methods.

This project focuses on evaluating and comparing the two distinct approaches for gesture classification: a classical image analysis pipeline and a CNN-based model. In Part 1, classical methods are used to identify features such as fingers and hand orientation, relying on deterministic algorithms and manually defined logic. Part 2 involves training a CNN model to classify gestures by automatically learning features from preprocessed images. The primary goal is to assess the performance of these approaches and determine how classical methods compare to the adaptive capabilities of CNNs for hand gesture recognition.

# Part 1: Implementing Vision Based Techniques to classify Hand Gestures

## Methodology:

To begin tackling this assignment, preprocessing of the images was necessary to prepare them for analysis. Initially, edge detection methods such as Sobel and Canny were applied to attempt to extract the edges and details of the hand and the fingers. However, this didn't work due to the low quality of the images, which are very blurry and pixelated. Even after applying many other techniques such as Gaussian filters and Prewitt operators to enhance the image quality, the edges of the fingers could not be clearly extracted.

After seeing this, a different approach was tested which was converting the images into binary format to simplify the detection process. The idea was to extract the edges by identifying the boundaries between the black and white regions in the binary image. However, generating a clean binary image also proved to be challenging. After experimenting with several methods, HSV color masking was used to successfully isolate the hand from the background and create a clear binary image for further processing.

The HSV (Hue, Saturation, Value) colour space was chosen because it distinguishes between information about colour (hue) and intensity (value), making it more robust to variations in lighting. By isolating the background using a specific range of blue (in this case, lower and upper thresholds of `[90, 50, 50]` and `[130, 255, 255]`, respectively), I could create a mask to remove the background. Using `cv2.bitwise_not` on the mask effectively inverted it, isolating the hand as the region of interest. The masked image was then converted to grayscale to reduce computational complexity, and finally, thresholding was applied to convert the grayscale image into a binary image, by setting the threshold to 1 and maximum 255, to identify the edge of the hand better. HSV colour masking accounts for variations in brightness and shadows that might affect simple grayscale thresholding for edge detection methods.

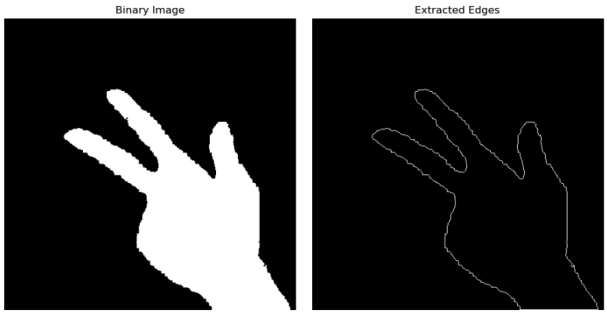



Figure 1: Shows the crisp binary hand gesture and the edges extracted

After successfully extracting the edges of the hand, the next step involved determining a method to classify hand gestures. This classification was based on identifying the number of open fingers and whether the thumb was open or closed. Initial research into existing approaches suggested using only the area of the hand for classification. However, after plotting histograms for different gesture classes, it was observed that the area varied

minimally between classes, making this method ineffective. Through extensive trial and error, a more reliable approach was developed.

Contour detection was applied to identify the boundaries of the features in the binary image, with the largest contour being selected to represent the hand. This decision was made because the largest contour typically corresponds to the main object of interest—in this case, the hand. The smallest convex shape that can contain every point on the contour is represented by the convex hull of the largest contour, which was then calculated. Convex hulls were used as they simplify the shape of the hand, making it easier to analyze.

Convexity defects were also utilized, which are geometric features that highlight deviations between the contour and its convex hull. These defects were particularly useful for identifying the fingers and the valleys between them, allowing for an effective classification of the number of fingers present in the hand gesture.

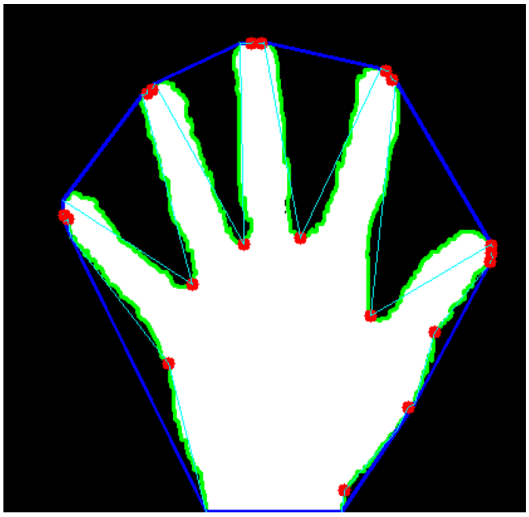


Figure 2: Visualisation of Convex defects and Largest Contour

To determine the centre of the palm, image moments were utilized. Image moments are a mathematical representation of a region's properties, and in this case, they were used to calculate the centroid of the hand's largest contour. This approach was selected as it provides an accurate method for identifying the palm's geometric center, which serves as a key reference point for further analysis of the hand gestures. To detect the number of open fingers and determine whether the thumb was open or closed, Euclidean distance and angle calculations were performed between the fingertips, valleys, and the palm center. The Euclidean distance allowed for precise measurement of spatial relationships, which helped to identify fingertips based on their distance from the palm.

Additionally, angles between points on the convex hull were calculated using trigonometric principles. A cosine-based formula was employed to calculate the angle formed at each defect, helping to filter out points that were not valid fingers. Specifically, points with an angle less than `90` degrees and a convexity defect depth greater than a threshold of `10000` were classified as fingertips. This method ensures that only prominent finger features were detected, reducing false positives caused by noise or irregularities in the contour.

A problem that was arising, as seen in figure 3, is that the thumb was being classified as a finger or the pinkie was being classified as the thumb due to them both being short fingers. To solve this issue, an additional condition was introduced to ensure that the thumb was classified correctly. First, the distance between the palm center and each fingertip was calculated, and a threshold was set such that any fingertip within a distance of less than `175` pixels from the palm was considered as a candidate for the thumb. Following this, an additional condition was applied to identify the thumb as the rightmost finger by ensuring its x-coordinate was greater than `320`. Since the test dataset only included images of the right hand, this assumption ensured correct identification of the thumb. These combined conditions effectively ruled out the pinkie from being misclassified as the thumb.

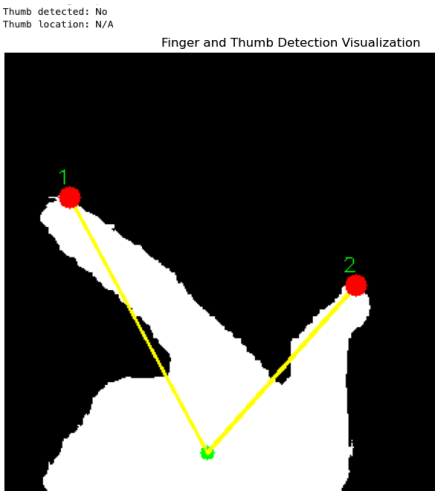



Figure 3: Misclassification of the thumb

Earlier it was pointed out that another issue that was arising was that a finger was being assigned 2 fingertips as seen in figure 4. This is due to the convexity defect detection algorithm which was identifying 2 closely spaced peaks on the same finger as separate fingertips resulting from noise or irregularities in the contour of the binary hand image. To improve the accuracy of fingertip detection, a merging mechanism was introduced to handle cases where two detected fingertip points were too close to each other. Fingertips separated by a Euclidean distance of less than 30 pixels, which was calculated to be the average distance between these misclassified fingertips, were considered to represent the same finger. To resolve this, the coordinates of these closely spaced points were averaged, and a new merged fingertip was created. This merging process ensured that only one representative fingertip was retained for each finger thus improving consistency in the hand gesture detection.

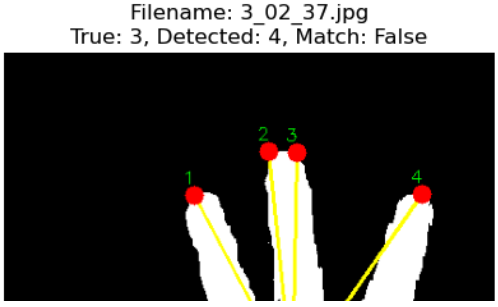
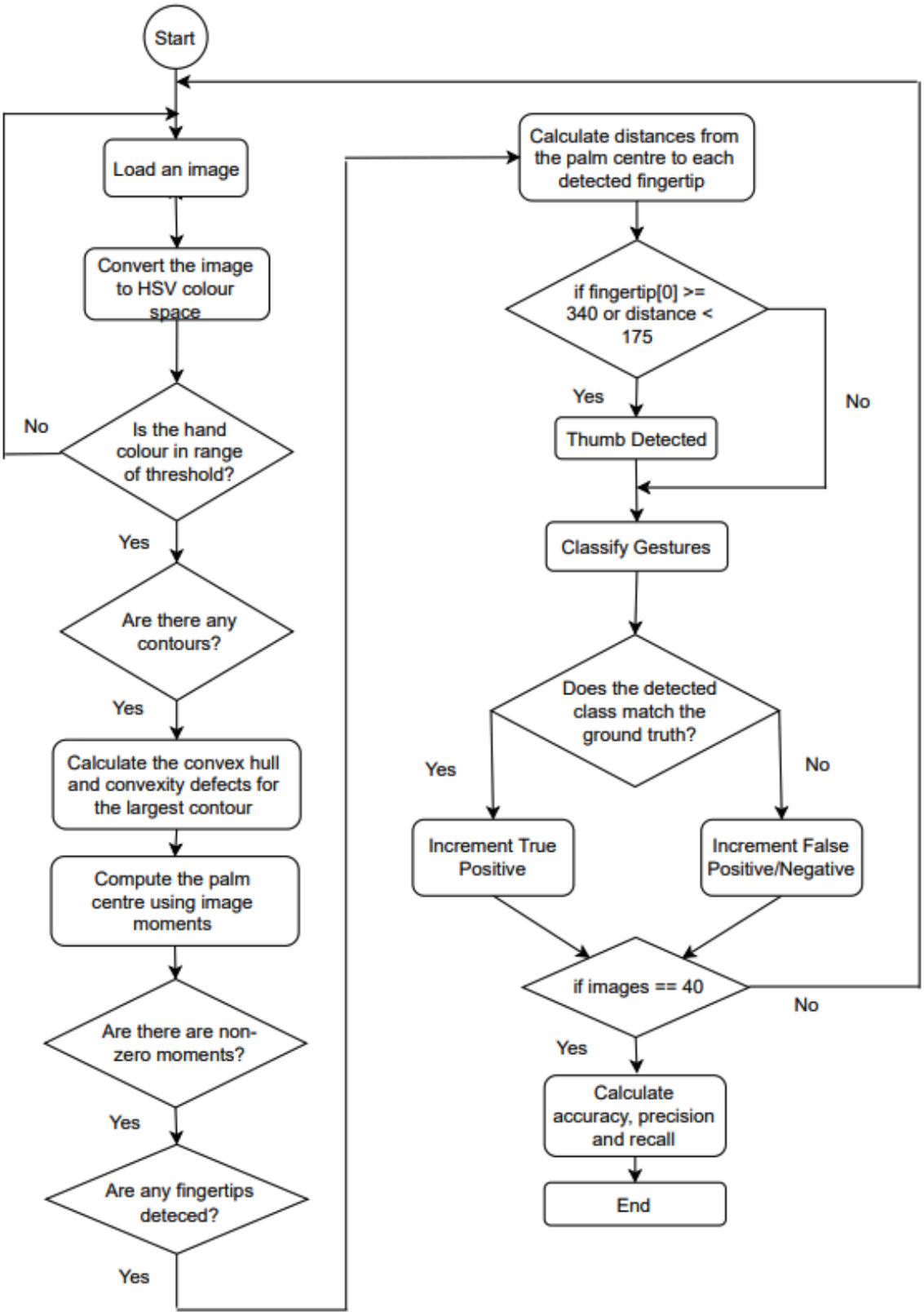



Figure 4: Image analysis for a finger with 2 fingertips detected

Flowchart for the Classical Computer Vision Techniques Algorithm:



**Results:**

Table 1 shows the results obtained:

| Class | Precision | Recall | True Positives | False Positives | False Negatives |
|---|---|---|---|---|---|
| 0 | 1.000000 | 1.00 | 4 | 0 | 0 |
| 1 | 1.000000 | 1.00 | 4 | 0 | 0 |
| 2 | 1.000000 | 1.00 | 4 | 0 | 0 |
| 3 | 1.000000 | 0.75 | 3 | 0 | 1 |
| 4 | 1.000000 | 0.50 | 2 | 0 | 2 |
| 5 | 1.000000 | 1.00 | 4 | 0 | 0 |
| 6 | 1.000000 | 0.75 | 3 | 0 | 1 |
| 7 | 0.800000 | 1.00 | 4 | 1 | 0 |
| 8 | 0.800000 | 1.00 | 4 | 1 | 0 |
| 9 | 0.666667 | 1.00 | 4 | 2 | 0 |
| Overall Accuracy | 0.900000 | NaN | 36 | 4 | 4 |

The table of results indicates that precision and recall are high for most classes, with an overall accuracy of 90%. However, some images from Class 3, Class 4, and Class 9 are being misclassified, as evidenced by the lower recall for Class 3 and the precision values for Class 4 (0.50) and Class 9 (0.67), which are affected by higher false positives, as shown in the confusion matrix. The confusion matrix highlights that misclassifications primarily occur between geometrically similar gestures, such as Class 4 being confused with Class 9 and Class 6 with Class 7. This suggests that the algorithm struggles with distinguishing subtle differences in finger positions for certain gestures.

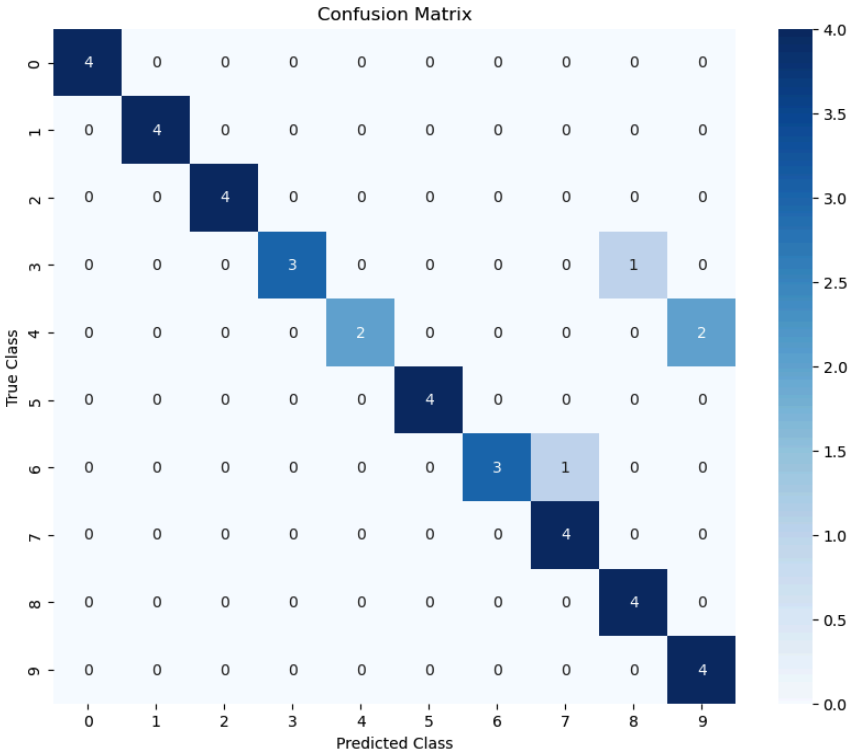



Figure 5: Confusion Matrix obtained from the results for better visualisation

The 4 images that aren't being classified correctly are the following;

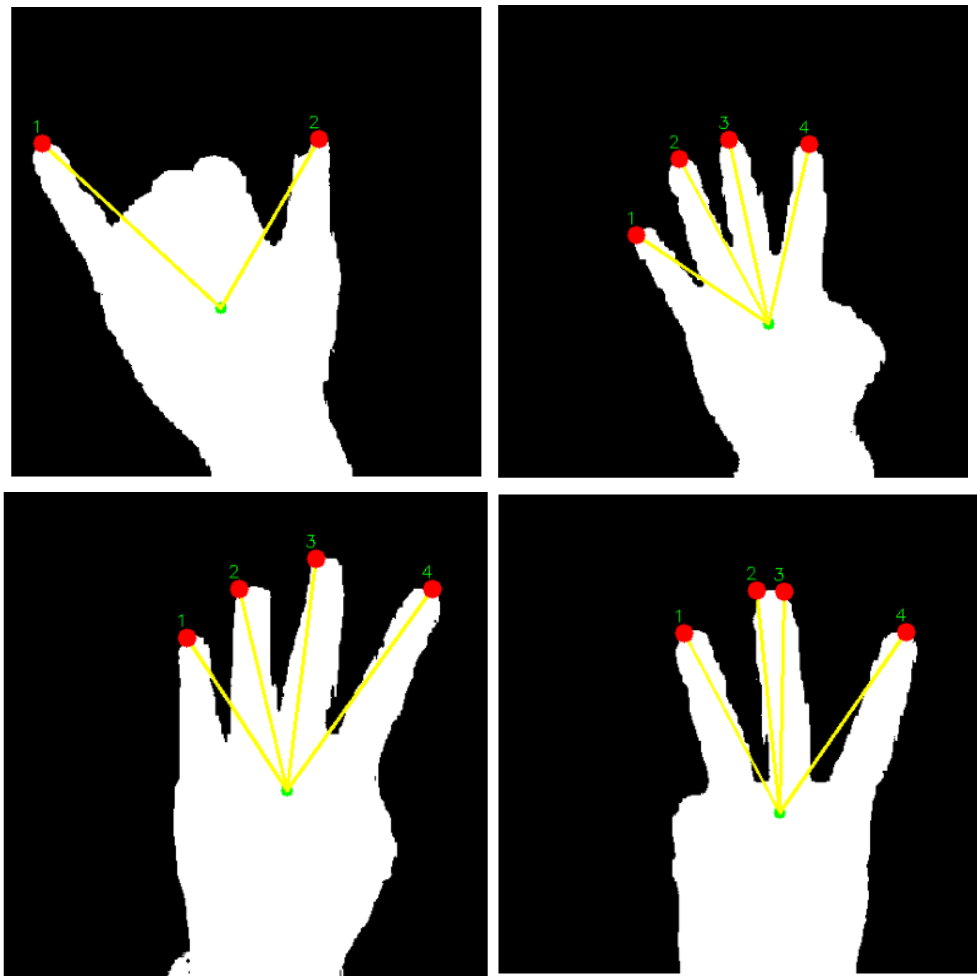



Figure 6: Shows all the misclassified images, top left Class 6, top right Class 4, bottom left Class 4 and Bottom Right Class 3

The class 6 hand gesture is being identified as class 7 due to the algorithm mistakenly identifying the pinkie as the index finger. This happens because the angle between the pinkie and the thumb is too similar to the angle typically observed between the index finger and the thumb. Since the algorithm heavily relies on geometric relationships to classify gestures, this overlap leads to the incorrect misclassification of the class.

The two, class 4 images are being classified as 9 because this algorithm is detecting the thumb. In the bottom left case, the index finger is too far to the right and beyond the 340 threshold. In the second image, the algorithm is either identifying a bulge in the palm contour as a finger or misclassifying the short index finger as the thumb. Tweaking the parameters like the distance thresholds and x-coordinates improved accuracy for specific cases but then reduced the overall accuracy by causing other images to be misclassified. This highlights the balance that must be struck between achieving precision for particular hand gestures and maintaining the algorithm's ability to generalize effectively across all cases.

The class 3 image being detected as a class 8 image, demonstrates that although 2 fingertips were detected on the same finger they are being merged into 1 because only 3 fingers were detected in the image. However the problem arises that the thumb is being classified as open which results in the class 8 image characterisation, hence the thumb detected logic needs to be refined but with the current algorithm doing so would result in other images being misclassified and reducing the overall accuracy, highlighting the limitations that come from classical computer vision techniques.

From these results, it is clear that the x-coordinate threshold of 340, while effective in some instances, has its limitations. This threshold is assuming that the thumb will always be positioned in a specific region of the image space, which is not always guaranteed. Variations in hand orientation, scaling, or misaligned captures are leading to situations where the thumb is misclassified or missed entirely. Furthermore, relying on the fixed distances threshold, of less than 175 pixels from the palm center, also introduces limitations as it assumes a uniform hand size and distance, which is not true in all cases especially if new and different hand gesture orientations images are introduced to the Train Dataset.

A big limitation of the algorithm implemented is how much it relies on the geometric assumptions and fixed thresholds for the angles and distances, which makes it difficult to adapt to variations. Improvements include using machine learning techniques [as seen in Part 2] since they eliminate the need for these fixed thresholds and make the model more flexible to changes. Using different input data such as adding depth of information or even using 3D hand models could also improve the accuracy by providing more detailed information about the hand's structure. Since, the poor quality of the images, including low resolution and noise, makes it harder to extract features accurately, which affects the overall performance of the system.

# Part 2: Implementing a Convolutional Neural Network to classify Hand Gestures

## Methodology:

The CNN model was constructed using separately loaded training and testing datasets. To preprocess the training data, the same methodology from Part 1 was applied, which involved converting the images to binary format but leaving the hand in grayscale for better feature identification. Each image was read, processed using HSV color masking to isolate the hand from the background, and then converted to grayscale. The grayscale image was resized to a fixed dimension of 128x128 pixels and normalized by dividing pixel values by 255. These preprocessing steps ensured consistency in the input data for the CNN. The processed images were saved as a 4D NumPy array with the shape (360, 128, 128, 1), where 360 represents the total number of samples, 128x128 are the dimensions of each image, and 1 denotes the grayscale channel.

The class labels were extracted from the folder names containing the images, converted to numerical values, and one-hot encoded for classification, resulting in a label array of shape (360, 10). The test dataset was preprocessed similarly to the training data to ensure consistency. Images were loaded from the specified test directory, with the first digit of each image filename used as the class label and the rest of the processing is the same as the training dataset.

For the next step, to improve the training dataset's diversity and lower the chance of overfitting, data augmentation was implemented. By introducing the *augment_image* function, multiple transformations were applied to each input image, simulating variations that the model might encounter. The augmentation techniques implemented are the following;

1. Rotation: Images are rotated by a random angle of either -15 or +15 degrees which simulates slight rotations in the orientation of the hand which occurs for different hand gestures of the same class.
2. Horizontal Flip: Images are flipped horizontally to account for variations in hand orientation, such as mirrored gestures when accounting for left or right hands.
3. Scaling: Images are scaled randomly between 90% and 110% of their original size, therefore helping the model learn to recognize gestures at varying distances from the camera.
4. Brightness Adjustment: to account for different lighting conditions, a random brightness factor between 0.8 and 1.2 is applied.
5. Gaussian Noise: To help the model classify noisy images better, gaussian noise, with a mean of 0 and standard deviation chosen randomly between 5 and 15, was added to the image.

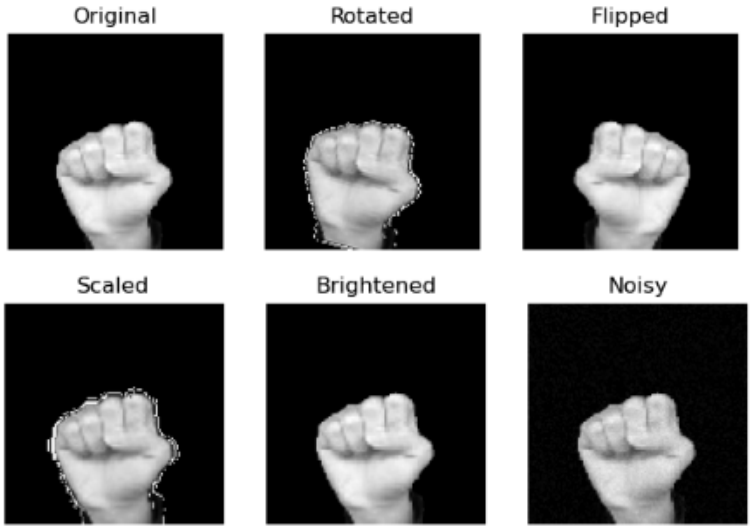



Figure 7: Augmented Images

By generating batches of augmented images, random samples from the training dataset were augmented, and one randomly chosen augmented version of each sample was used for training which ensured continuous diversity in the training data. Next came the CNN model which consisted of 3 convolutional blocks, followed by a fully connected (dense) layer as seen in figure 8. After experimenting with different layers and parameters for the CNN model, the following structure was adopted from [4] which made a big difference in the accuracy. The `MaxPooling2D` layer was adopted to reduce the spatial dimensions and it proved to be very effective. Each convolutional block included the following;

- Each block had a `Conv2D` layer with ReLU activation and filter sizes increasing from 32 in the first block to 64 to 128 in the third block, all using a kernel size of (3,3).
- A `MaxPooling2D` layer with a pool size of (2, 2) was applied after each convolutional layer to reduce spatial dimensions and computational complexity.
- Dropout was applied to prevent overfitting by randomly deactivating neurons during training. The dropout rates were progressively increased from 0.3 in the first block to 0.4 in the second and third blocks.

After the convolutional layers, the following were applied;

- The feature maps were transformed into a 1D vector using flatten layers so they could be fed into the dense layers.
- High-level features were learnt using a dense layer with 256 units and ReLU activation. For additional regularisation, a second Dropout layer with a rate of 0.5 was added.
- Class probabilities were provided for the output layer using a final dense layer with 10 units and a softmax activation function.

Following the construction of the CNN model, the compilation was done by using the '*cross-entropy'* for the loss function, for the optimizer '*RMSprop'* was chosen since it adapts the learning rate for each parameter based on recent gradients and finally the '*accuracy'* metric was used to monitor the model's performance.

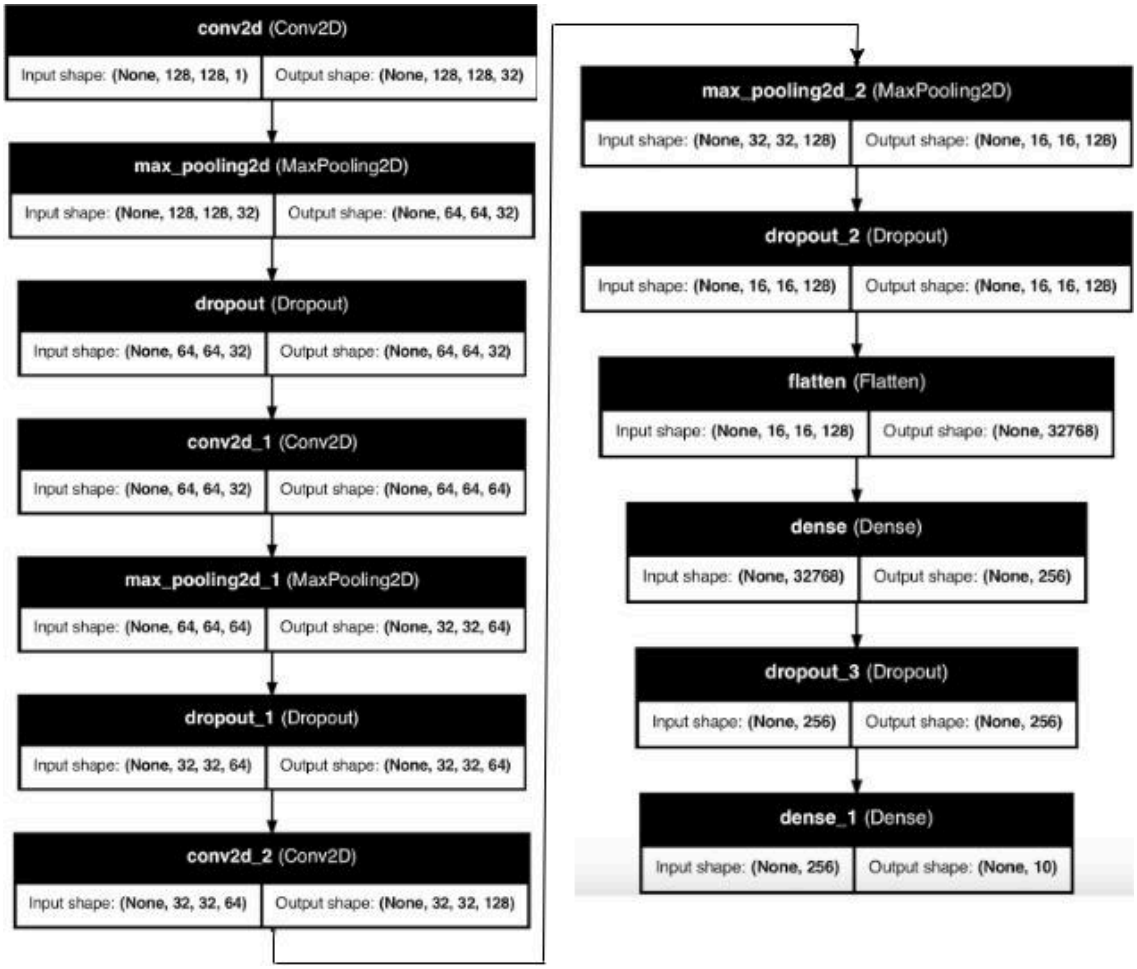



Figure 8: Diagram of the CNN architecture

The model was trained for a different number of epochs but 50 proved to give the best results with a batch size of 32. An adaptive learning rate algorithm was implemented to help refine the model's performance by employing the `ReduceLROnPlateau` callback to reduce the learning rate by a factor of 0.5 if the validation loss plateaued for 3 epochs, with a minimum learning rate of 1 x 10^-6. After training the CNN, the model's performance was evaluated on the test set.

**Results:**

| Class | Precision | Recall | True Positives | False Positives | False Negatives |
|---|---|---|---|---|---|
| 0 | 1.000 | 1.00 | 4 | 0 | 0 |
| 1 | 1.000 | 1.00 | 4 | 0 | 0 |
| 2 | 1.000 | 0.75 | 3 | 0 | 1 |
| 3 | 1.000 | 1.00 | 4 | 0 | 0 |
| 4 | 0.800 | 1.00 | 4 | 1 | 0 |
| 5 | 1.000 | 1.00 | 4 | 0 | 0 |
| 6 | 1.000 | 1.00 | 4 | 0 | 0 |
| 7 | 1.000 | 1.00 | 4 | 0 | 0 |
| 8 | 1.000 | 1.00 | 4 | 0 | 0 |
| 9 | 1.000 | 1.00 | 4 | 0 | 0 |
| Overall Accuracy | 0.975 | NaN | 39 | 1 | 1 |

From the above results, one can see that the CNN model achieves very high accuracy of 97.5% by only misclassifying 1 image. Class 2 has a lower recall of 0.75, which corresponds to the single misclassified image seen in the confusion matrix which was classified as Class 4 instead of Class 2.

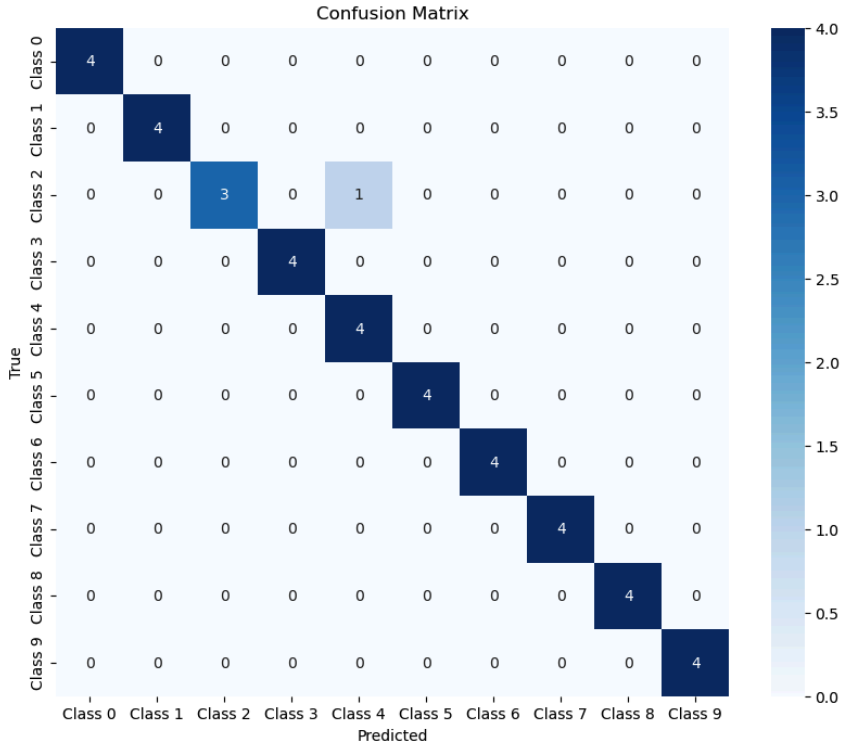



Figure 9: Confusion Matrix obtained from the results for better visualisation

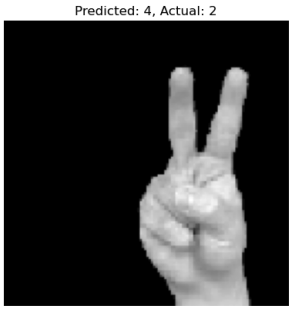



Figure 10: Misclassified Image

The misclassified image, Class 2 predicted as Class 4, shows a hand gesture with two raised fingers. A reason for this misclassified image is that the CNN may have interpreted additional noise or subtle contour details in the binary image as resembling four fingers. Which highlights potential limitations in the model's ability to generalize when gestures have overlapping features, such as variations in finger positions or angles.

Although the models were tuned extensively and various parameters were experimented with to achieve the best possible results, there are still limitations that can be addressed. Additional improvements include introducing more variations of Class 2 and Class 4 gestures in the augmentation section, involving rotations, added noise, or brightness

adjustments. This would allow the CNN to generalize better and recognize subtle differences between similar gestures. Furthermore, adding additional convolutional layers to the architecture may help capture finer details within the images. Exploring deeper architectures could also significantly improve performance by enabling the model to learn more complex and hierarchical features.

After all the results were taken, an error in the algorithm was recognised where the training data wasn't being split into training and validation data and this was seen as an opportunity to see the real impact of validation data on the algorithm. When splitting the data to 80% training and 20% validation, the previous algorithm didn't perform so well since the validation loss was consistently lower than the training loss. After some research it was found that this could be due to the dropout function which if active during the training adds noise or constraints. The code was then modified, by removing the augmentation function completely and reducing the number of epochs to 30 to achieve better results. After some tuning, an accuracy of 88% was achieved which isn't as good as the algorithm without the validation test data. The Training and Validation loss graph and training and Validation Accuracy graph were obtained as shown in Figure 11.

When the data was not split into training and validation sets, the model had direct access to the entire dataset, allowing it to "memorize" patterns and details. This resulted in fewer misclassifications during evaluation since the model essentially overfitted to the training data. However, this approach lacked an independent validation set to assess overfitting or generalization. Once a validation split was introduced, it reduced the amount of data available for training which forced the model to generalize better to unseen validation data, preventing it from simply memorizing the dataset. While this improved the model's ability to generalize, the reduced training data made it slightly harder for the model to learn every detail, leading to a slight increase in misclassifications on the test set.

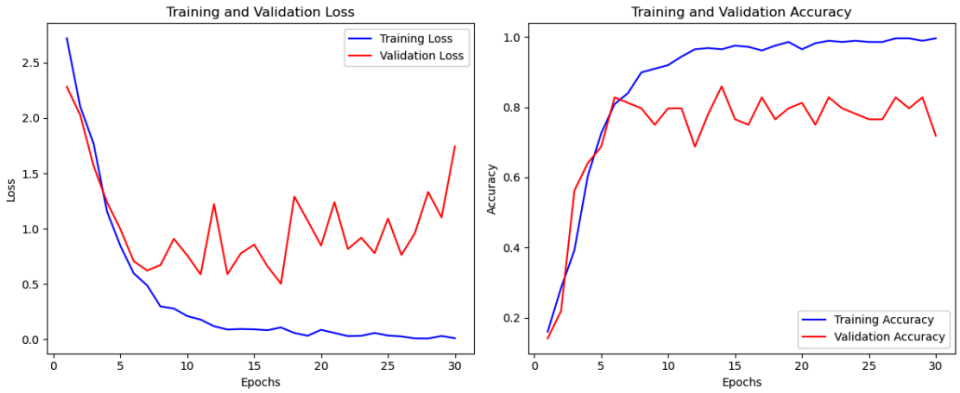



Figure 11: Training and Validation Loss and Accuracy Plots

From the plots one can see that the training loss decreases steadily, indicating that the model is learning well on the training data. However, the validation loss decreases initially but starts fluctuating and increases towards the end, suggesting overfitting as the model performs worse on unseen data. To improve the validation loss, the regularization and early stopping should be adjusted to mitigate the overfitting occurring.

## Conclusion:

The results of this project demonstrate the strengths and weaknesses of both traditional image analysis and CNN-based approaches to hand gesture classification. While the CNN model achieved better overall accuracy, the importance of validation data became apparent during the experimentation process. Initially, the lack of a validation split allowed the model to overfit to the training data, resulting in fewer misclassifications but providing an unrealistic measure of performance. Introducing a validation split forced the model to generalize better, but it also revealed overfitting issues and increased the number of misclassified images, as the training data was reduced.

The validation loss fluctuating and eventually increasing beyond the training loss further confirmed the overfitting issue. Adjustments to the training process, such as removing augmentation, reducing epochs, and tweaking hyperparameters, helped improve the balance between training and validation performance. However, this revealed the trade-offs between overfitting and generalization, with a slight drop in performance on the test set compared to the initial overfitted model.

These findings highlight the challenges of achieving robust generalization in CNNs, particularly with limited data. While CNNs excel at learning complex features and require less manual effort compared to traditional methods, their sensitivity to overfitting emphasizes the importance of proper validation and regularization techniques. Future work could explore combining the strengths of traditional methods with CNNs to better handle subtle variations in hand gestures and achieve even higher accuracy.

## References:

[1] Mitra, S., & Acharya, T. (2007). Gesture recognition: A survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(3), 311-324.

[2] Pavlovic, V. I., Sharma, R., & Huang, T. S. (1997). Visual interpretation of hand gestures for human-computer interaction: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7), 677-695.

[3] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25, 1097-1105.

[4] HarshJuly12. (n.d.). *Hand Gesture Recognition Using Convolutional Neural Networks (CNN)*. GitHub. Retrieved December 10, 2024, from https://github.com/harshjuly12/Hand-Gesture-Recognition-Using-Convolutional-Neural-Networks-CNN/blob/main/HandGestureRecognitionUsingCNN.ipynb