**American Sign Language Classification using CNNs:**

**A Comparative Study**

Pranav Neupane1, Samrat Kumar Adhikari2\*, Shyama Mainali3, Utsarga Regmi4, Pralhad Chapagain5

*1Department of Computer and Electronics Engineering, Kantipur Engineering College, Dhapakhel, Lalitpur, Nepal,*

*neupanepranav27@gmail.com*

*2Department of Computer and Electronics Engineering, Kantipur Engineering College, Dhapakhel, Lalitpur, Nepal,*

*samratmetaladhikari@gmail.com*

*3Department of Computer and Electronics Engineering, Kantipur Engineering College, Dhapakhel, Lalitpur, Nepal,*

*mainalishyama@gmail.com*

*4Department of Computer and Electronics Engineering, Kantipur Engineering College, Dhapakhel, Lalitpur, Nepal,*

*regmiutsarga7@gmail.com*

*5Department of Computer and Electronics Engineering, Kantipur Engineering College, Dhapakhel, Lalitpur, Nepal,*

*pralhadchapagain@kec.edu.np*

|  |
| --- |
| **Abstract**  American Sign Language (ASL) classification plays a crucial role in facilitating communication for individuals with hearing impairments. Traditional methods rely heavily on manual interpretation, which can be time-consuming and error prone. Inspired by the success of deep learning techniques in image processing, the paper explores the application of Convolutional Neural Networks (CNNs) for ASL classification. The paper presents a custom CNN architecture tailored specifically for this task and investigates the effectiveness of transfer learning by leveraging four pre-trained models: VGG16, InceptionV3, ResNet50, and DenseNet121. A comparative analysis of these architectures has been presented in this paper. The experimental results show that with a testing accuracy of 99.93%, customized CNN model outperformed other models when provided with real time images. Consequently, it is concluded that customized CNN outshines other models in accurately classifying sign languages.  *Keywords*: American Sign Language, Deep Learning, Convolutional Neural Network, Transfer Learning, Image Classification |

1. **Introduction**

Communication is crucial for everyone but individuals with speech and hearing challenges often struggle to express themselves and understand others, leading to feelings of isolation. Sign language is manual communication commonly used by people who are deaf. Sign language is not universal, people who are deaf from different countries speak different sign languages.[1] To address this issue, researchers have developed hand sign recognition systems, enabling communication through hand gestures.

. Initially relying on basic features like color and shape for sign and gesture detection, these systems have evolved significantly since their inception in the late 20th century. However, advancements in computer vision and deep learning, particularly with CNNs, have propelled hand sign recognition forward. CNNs excel at learning patterns from visual data, enabling them to interpret hand gestures accurately by analyzing both shape and movement. Unlike older methods that were manual and less adaptable, CNNs automatically improve with training, making hand sign recognition more efficient. These advancements hold promise for enhancing the quality of life for individuals with speech and hearing impairments by providing them with more accessible means of communication. This paper presents a comprehensive study of a custom CNN architecture and four pre-trained CNN models. The primary objective of this study is to evaluate the performance of these models and determine the model with the highest accuracy in classifying hand signs from the video frame from a live camera.

1. **Related Works**

The research on Hand Sign Recognition utilizing CNN employed a tailored CNN model achieving a 75% accuracy in classifying hand signs, and for background elimination, the HSV color space was utilized to ensure the model's effectiveness across diverse environments.[2] HSV (Hue, Saturation, Value) color space separates color information from brightness, enabling the extraction of the hand region based on its distinctive color characteristics, such as skin tone, while disregarding variations in lighting conditions.[3] A paper published in 2021 illustrated the use of Mediapipe library to isolate hand region based on the hand landmarks from an image. Mediapipe provides a framework for machine learning-based hand gesture recognition, enabling the extraction of hand regions from images through sophisticated computer vision techniques, thereby enhancing interactivity and usability in applications like user guides.[4] Another paper published in 2018 presents a method for hand gesture recognition that combines background subtraction with convexity defect detection. By eliminating irrelevant information through background subtraction and utilizing convexity defects for feature extraction, the proposed approach ensures reliable gesture recognition invariant to translation and rotation.[5] Transfer learning is gaining popularity in image classification, where a pre-trained neural network is repurposed for a new task. This method leverages the learned features of the pre-trained model, typically trained on a large dataset like ImageNet, and fine-tunes it for a specific classification task with a smaller dataset. By utilizing this existing knowledge, the model can adapt more quickly and effectively to the new task, leading to improved performance and reduced training time compared to starting training from scratch. A research paper titled “Sign language recognition: A comparative analysis of deep learning models” published in 2022 demonstrated a comparative study of a customized CNN and a VGG16 models, later being a transfer learning model. The paper concluded that VGG16 was better where it delivered an accuracy of 99.56% followed by customized CNN with an accuracy of 99.38%.[6] Hand Gesture Recognition (HGR) dataset was used in another paper where InceptionV3 and EfficientNet-B0 models were implemented. In this experimental study, InceptionV3 model achieved 90% accuracy with 0.93% precision, 0.91% recall, and 0.90% f1-score, respectively whereas EfficientNet-B0 achieved 99% accuracy with 0.98%, 0.97%, 0.98%, precision, recall, and f1-score respectively.[7] 2-level ResNet50 was used in a paper title “Sign Language Recognition Using ResNet50 Deep Neural Network Architecture” where the model achieved an accuracy of 99.03% on 12,048 test images.[8] DenseNet, an another CNN architecture has become widely popular and are extensively used in image processing tasks. A research paper published in 2021 performed comparative study of ResNet50 and DenseNet121 on American sign language dataset. The dataset was split into training and testing set in the ratio 80:20. ResNet50 achieved an accuracy of 0.999913 with recall, precision, and f1-score of 0.998966, 0.998958, and 0.999913 respectively. The same train-test ratio was used to train and test the DenseNet121 model where it achieved an accuracy of 0.998872 with recall. Precision, and f1-score of 0.987116, 0.986458 and 0.998872.[9] An ASL dataset sourced from the Modified National Institute of Standards and Technology (MNIST) database was used for a research paper titled “American Sign Language Recognition Based on Transfer Learning Algorithms” where detailed comparison between VGG16, ResNet50, MobileNetV2, InceptionV3 and customized CNN. The paper illustrated that VGG16 and InceptionV3 stand out among the stated models where VGG16 and InceptionV3 performed with an accuracy of 0.95 and 0.96 respectively.[10]

Despite the significant advancements made in the field, there remains a notable research gap that needs addressing. The proposed research titled “**American Sign Language Classification: A Comparative Study**” aims to fill this gap by investigating the efficiency of transfer learning in ASL recognition, utilizing pre-trained models to enhance accuracy. This study seeks to contribute to the field by examining how transfer learning can improve ASL recognition with various pretrained and customized models and comparing their effectiveness in testing set and with real time images.

1. **Methodology**

The dataset used for this research was acquired from Kaggle, a renowned website for data science and machine learning notebooks and datasets. The model was trained using Google Colab due to its provision of free GPU services, offering cost-effective access to accelerated computing resources.

* 1. ***Dataset Description***

The acquired dataset has a training set and a testing set subfolders. The training set subfolder was used for training, validation, and testing. There are 29 classes in the training folder: 3 classes for ‘del’, ‘space’, and ‘nothing’, and 26 classes for the English alphabet, each with 3000 images, making a total of 81000. However, ‘del’ and ‘nothing’ classes were removed for this research. The training subfolder was split into training, validation and testing set in the ratio 68:17:15.



Figure 1: Image samples from each class

Figure 1 depicts images of hand signs across different classes, showcasing variations in brightness and distance from the camera. Additionally, similarities between certain signs, such as V and K, S and E, and X and I, are observed.

A graph of purple lines

Description automatically generated with medium confidence

Figure 2: Distribution of images in each class

The above figure indicates that each of the 27 classes comprises 3000 images, ensuring a balanced dataset. Balanced datasets are essential as they mitigate the risk of overfitting and bias towards classes with large number of images.

* 1. ***Preprocessing***

Images in the training set were reduced to 128x128x3 to minimize computational complexity while maintaining the necessary information. By standardizing pixel values between 0 and 1, normalization was used to improve data integrity and decrease redundancy.

Preprocessing techniques were applied to live images taken from a camera to classify hand signs in real time. The Hand tracking module from Mediapipe was used to extract the hand region from every frame. Furthermore, hand contours were utilized to generate background subtraction masks, guaranteeing adaptability to different environments. The resulting images were resized to 128x128x3, and pixel values underwent normalization to match the training set.

* + 1. ***Mediapipe library for hand region extraction***

Mediapipe is an open-source framework developed by Google which is used to build pipelines that perform computer vision inference over arbitrary sensory data such as video or audio. Mediapipe is used to detect hand landmarks with the help of its hand tracking module. The hand tracking model detects 21 key points on the hand. These key points are used to track the hand and crop the hand region from the frame. The model was trained on more than 30k real world images over varying backgrounds.[11]

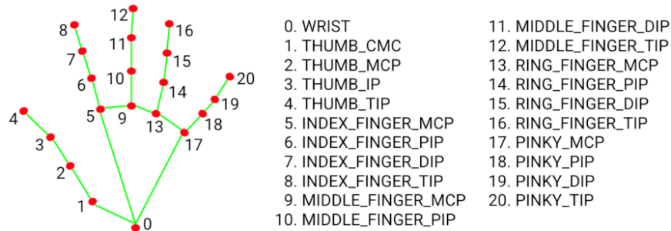


Figure 3: Illustration of hand landmarks using Mediapipe

* + 1. ***Background subtraction using hand contours***

Contours are the continuous line formed by connecting points along the boundary of the hand, determined by its color or intensity. These contours serve as a delineation of the hand's shape. They can be employed as a mask for background subtraction by isolating the hand from its surroundings. By utilizing contours as a guide, background subtraction algorithms can accurately segment the hand from the background, facilitating tasks such as gesture recognition or object tracking in computer vision applications.

The mask derived from contours can be seamlessly overlaid onto the cropped hand image, effectively eliminating the background. This process ensures a clean separation between the hand and its surroundings, enhancing the focus on the hand itself for further analysis or visualization purposes.

A collage of a person holding a fist

Description automatically generated

Figure 3: Hand region segregation using Mediapipe and hand contours

***3.3 Model Selection***

Convolutional Neural Networks (CNNs) stand out as the preferred choice for image processing tasks. They are a particular kind of feed-forward neural network that are well known for their capacity to learn feature representations on their own by optimizing filters or kernels. The basic architecture of a CNN consists of multiple layers, including convolutional layers, pooling layers, and fully connected layers. In the convolutional layers, the network applies a set of filters to the input image to extract features such as edges, corners, and textures. The pooling layers are used to reduce the spatial dimensions of the feature maps, while the fully connected layers are used to classify the input image based on the extracted features. CNNs are trained using large datasets of labeled images, and they use backpropagation to adjust the weights of the network to minimize the error between the predicted and actual labels. The training process involves several iterations of forward and backward propagation, where the network learns to recognize patterns and features in the input images. Essentially, CNNs are very good at extracting complex patterns and characteristics from images, which makes them very useful for a variety of tasks like object identification, image segmentation, and image classification.[12]

A diagram of a diagram of a process

Description automatically generated with medium confidence

Figure 4: CNN architecture

***3.3.1 Customized CNN***

Customized CNNs are tailored neural network architectures designed to address specific tasks or datasets. These networks are often modified by adjusting the number of layers, the types of layers used, or the layer parameters to improve performance on a particular problem. Customization can involve adding, removing, or altering layers such as convolutional, pooling, or fully connected layers. Additionally, the network's hyperparameters, such as learning rate, batch size, and regularization, may be adjusted to optimize performance. Customized CNNs are widely used in various applications, including image classification, object detection, and facial expression recognition, to achieve superior results on specific tasks or datasets.

***3.3.2 VGG16***

VGG16 is a 16-layer deep convolutional neural network (CNN) architecture, proposed by the Visual Geometry Group at the University of Oxford. The network consists of 13 convolutional layers and 3 fully connected layers. The input size is 224x224x3 (RGB images). The convolutional layers use 3x3 filters with a stride of 1 and padding to maintain the spatial dimensions. The network has a total of approximately 138 million parameters. The VGG16 architecture is known for its simplicity and effectiveness.[13]

A table of information

Description automatically generated with medium confidence

Figure 8: Detailed VGG16 architecture

***3.3.3 InceptionV3***

InceptionV3 is an advanced neural network architecture developed by researchers at Google. With a total of 48 layers, this complex network is intricately designed to analyze images and extract features in a hierarchical manner. The standout feature of InceptionV3 lies in its unique Inception modules, which redefine feature extraction by utilizing filters of varying sizes, from 1x1 to 5x5, in combination with strategic pooling techniques.Thanks to its comprehensive design and sophisticated feature extraction methods, InceptionV3 achieves exceptional accuracy in tasks such as image classification and various computer vision applications.

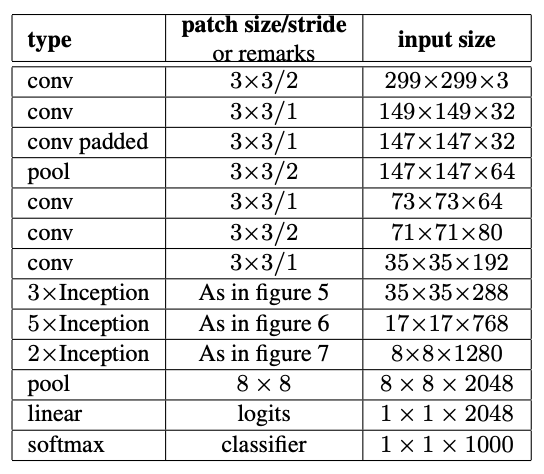
****

Figure 10: Detailed InceptionV3 architecture [14]

***3.3.4 ResNet50***

InceptionV3 ResNet50 is a neural network created by Microsoft researchers. It is designed especially for applications such as image recognition. It is a 50-layer convolutional neural network consisting of 48 convolutional layers, 1 max pooling layer, and 1 average pooling layer. Residual neural networks are a type of artificial neural network (ANN) that forms networks by stacking residual blocks.[15]

A table with numbers and symbols

Description automatically generated with medium confidence

Figure 12: Detailed ResNet architecture

***3.3.5 DenseNet121***

DenseNet121 is a robust neural network architecture that stands out for its dense connectivity pattern and is highly effective in tasks like image recognition. Unlike traditional networks where each layer is connected only to the next layer, DenseNet121 connects each layer to every other layer in a feed-forward fashion. This dense connectivity promotes feature reuse, enhances gradient flow, and combats the vanishing gradient problem, leading to more efficient training and improved accuracy.[16]

A white sheet with black text

Description automatically generated

Figure 14: Detailed DenseNet architecture

***3.4 Evaluation Metrics***

After constructing the system model, it undergoes training with the standardized training dataset. Following this, the model's performance is validated by testing it with the testing dataset. The effectiveness of the system's performance is evaluated based on metrics such as Accuracy, Precision, F1 score, and Recall. The confusion matrix is a table used to assess a classification model's performance. It provides a matrix-format summary of the model's predictions based on a dataset.

***3.4.1 Accuracy***

After Accuracy of a model is defined as the ratio of true positives and true negatives to all positive and negative observations.

Accuracy =

***3.4.2 Precision***

Precision of a model is defined as the percentage of labels that were correctly predicted positively.

Precision =

***3.4.3 Recall***

Recall or sensitivity of a model is defined as the ratio of true positives to all the positive instances.

Recall =

***3.4.3 F1-score***

F1-score is the metric that calculates the harmonic mean of precision and recall.

F1-score =

***3.5 Experimental results***

Table 1: Hyperparameters used

|  |  |
| --- | --- |
| Parameter | Value |
| Epoch | 30 |
| Batch size | 32 |
| Optimizer | Adam |
| Learning rate | 0.001 |

***3.5.1 Customized CNN***

The accuracy of the model for testing data is: 99.93%

The loss of the model for testing data is: 0.0023

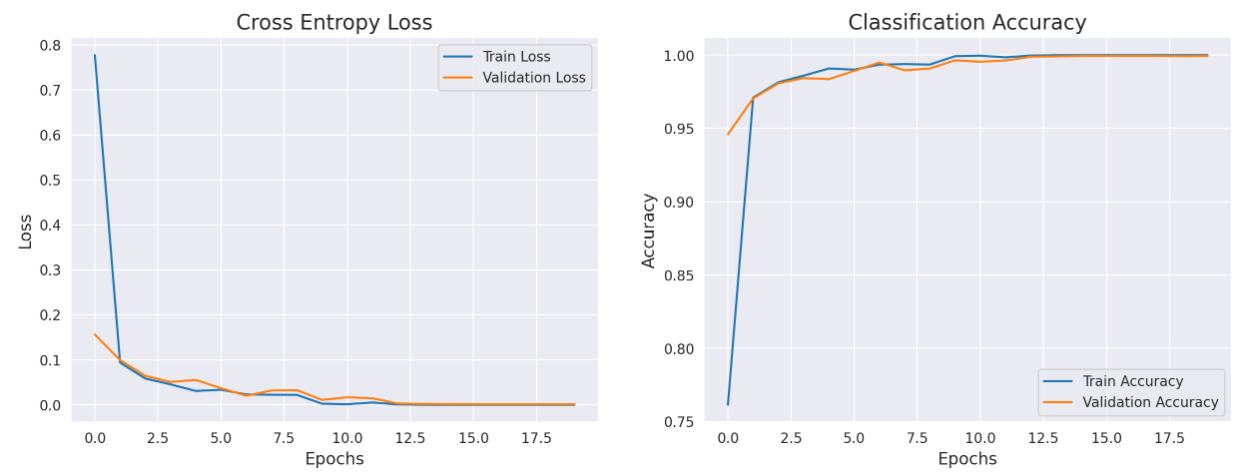


Figure 17: Training and validation set loss and accuracy for customized CNN

***3.5.2 VGG16***

The accuracy of the model for testing data is: 99.34%

The loss of the model for testing data is: 0.035

A graph of loss and loss

Description automatically generated

Figure 17: Training and validation set loss and accuracy for VGG16

***3.5.3 InceptionV3***

The accuracy of the model for testing data is: 81.62%

The loss of the model for testing data is: 0.56

A graph of a loss and a loss

Description automatically generated with medium confidence

Fig 18: Training and validation set loss and accuracy for InceptionV3

***3.5.4 ResNet50***

The accuracy of the model for testing data is: 99.70%

The loss of the model for testing data is: 0.008

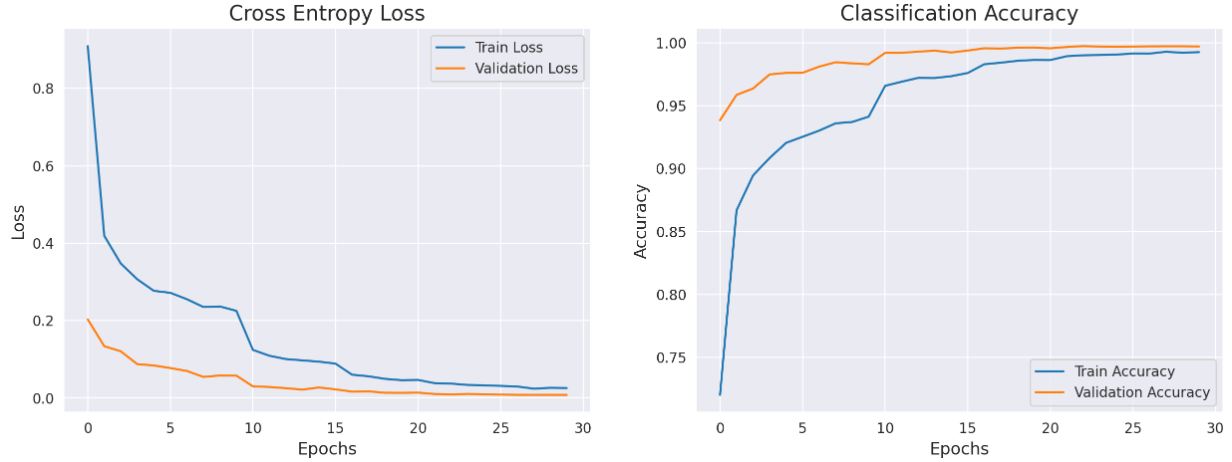


Fig 19: Training and validation set loss and accuracy for ResNet50

***3.5.4 DenseNet121***

The accuracy of the model for testing data is: 99.80%

The loss of the model for testing data is: 0.006

A graph of loss and loss

Description automatically generated

Fig 20: Training and validation set loss and accuracy for DenseNet121

***3.5 Model comparison***

Table 2: Model performance comparison on test set

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | Accuracy (%) | Recall | Precision | F1-score |
| Cust. CNN | 99.93% | 1.00 | 1.00 | 1.00 |
| VGG16 | 99.34% | 0.99 | 0.99 | 0.99 |
| InceptionV3 | 81.62% | 0.87 | 0.82 | 0.81 |
| ResNet50V2 | 99.70% | 1.00 | 1.00 | 1.00 |
| DenseNet121 | 99.80% | 1.00 | 1.00 | 1.00 |

The table above presents the performance metrics of various convolutional neural network (CNN) models on a classification task. Each model, including Custom CNN, VGG16, InceptionV3, ResNet50V2, and DenseNet121, is evaluated based on accuracy, recall, precision, and F1-score. Notably, Custom CNN achieves outstanding results across all metrics, with nearly perfect scores close to 1.00, indicating exceptional performance in correctly classifying instances. VGG16 also demonstrates high accuracy and strong performance across all metrics, while ResNet50V2 and DenseNet121 exhibit similarly excellent results. However, InceptionV3 falls short in comparison, with lower accuracy and performance metrics overall, albeit still performing relatively well. These findings suggest that Custom CNN, VGG16, ResNet50V2, and DenseNet121 are well-suited for the classification task, while InceptionV3 may require further optimization or may be less suitable for this specific dataset or task.

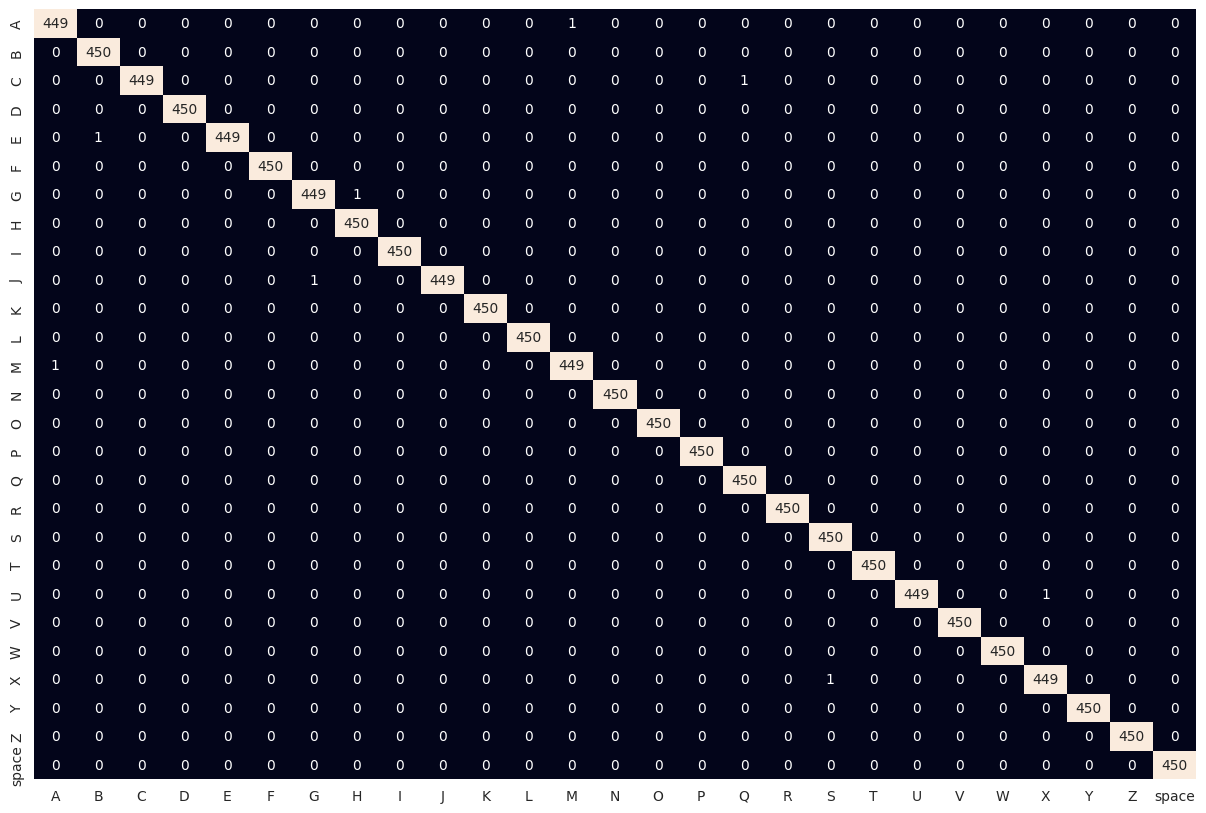


Figure 22: Confusion Matrix for each class in testing set using customized CNN

1. **Conclusion** **and Future Enhancements**

In In conclusion, the research on American Sign Language (ASL) recognition involved training several models, including a customized CNN, VGG16, InceptionV3, ResNet50, and DenseNet121. Through rigorous experimentation, it is found that the customized CNN exhibited superior performance in the testing dataset. This underscores the robustness of customized CNN in effectively recognizing ASL gestures in diverse environments. Moving forward, the study underscores the importance of selecting appropriate models tailored to the unique challenges of ASL recognition, paving the way for the development of more accurate and reliable ASL recognition systems. Furthermore, the integration of background subtraction using Mediapipe and hand contouring significantly contributed to our research findings. These techniques proved instrumental in mitigating environmental variations, enhancing the robustness of our ASL recognition system across diverse settings. Leveraging Mediapipe for background subtraction and hand contour extraction allowed for more accurate localization of hand gestures, thereby improving the overall performance and reliability of our ASL recognition models. Additionally, further exploration into advanced techniques such as data augmentation and model ensemble methods could offer avenues for even greater performance improvements in ASL recognition technology.

The current model exhibits occasional misclassification of images in real-time scenarios. However, there are promising avenues for improving its accuracy. By fine-tuning hyperparameters and adjusting the number of dense layers following the customized CNN architecture, the model's performance can potentially be enhanced, ensuring more precise recognition of American Sign Language (ASL) gestures. Additionally, optimizing computational efficiency is essential for real-time applications. Addressing the potential slowness of the present model in certain environments could involve refining model architecture, leveraging hardware acceleration, or exploring lightweight alternatives to accommodate faster inference speeds without compromising accuracy. These enhancements hold the potential to elevate the effectiveness and usability of the ASL recognition system, fostering greater accessibility and inclusivity for individuals with hearing impairments.

**Acknowledgement**

The authors would like to extend their heartfelt gratitude to the Department of Computer and Electronics at Kantipur Engineering College for their unwavering support and resources, which have been invaluable throughout the duration of this research endeavor. The authors are deeply grateful for their contributions, without which this research would not have been possible.

**References**

|  |  |
| --- | --- |
| [1] | “What is sign language? | AccessComputing,” www.washington.edu. https://www.washington.edu/accesscomputing/what-sign-language. |
| [2] | D. Bhavana, K. K. Kumar, M. B. Chandra, P. V. S. K. Bhargav, D. J. Sanjanaa, and G. M. Gopi, ‘Hand sign recognition using CNN’, *International Journal of Performability Engineering*, vol. 17, no. 3, p. 314, 2021. |
| [3] | U. Noreen, M. Jamil, and N. Ahmad, ‘Hand detection using HSV model’, *Hand*, vol. 6, no. 12, 2015. |
| [4] | M. Harris, A. S. Agoes, and Others, ‘Applying hand gesture recognition for user guide application using MediaPipe’, *in* *2nd International Seminar of Science and Applied* Technology (ISSAT 2021), 2021, pp. 101–108. |
| [5] | S. C. Mesbahi, M. A. Mahraz, J. Riffi, and H. Tairi, ‘Hand gesture recognition based on convexity approach and background subtraction’, *in 2018 International Conference on Intelligent Systems and Computer Vision (ISCV),* 2018, pp. 1–5. |
| [6] | A. Premkumar et al., ‘Sign language recognition: A comparative analysis of deep learning models’, *in Inventive Computation and Information Technologies: Proceedings of ICICIT 2021*, Springer, 2022, pp. 1–13. |
| [7] | A. Hussain, S. Ul Amin, M. Fayaz, and S. Seo, ‘An Efficient and Robust Hand Gesture Recognition System of Sign Language Employing Finetuned Inception-V3 and Efficientnet-B0 Network’, *Computer Systems Science & Engineering*, vol. 46, no. 3, 2023. |
| [8] | P. Rathi, R. Kuwar Gupta, S. Agarwal, and A. Shukla, ‘Sign language recognition using resnet50 deep neural network architecture’, *in 5th International Conference on Next Generation Computing Technologies (NGCT-2019*), 2020. |
| [9] | A. D. Marjusalinah, S. Samsuryadi, and M. A. Buchari, ‘Classification of finger spelling American sign language using convolutional neural network’, *Computer Engineering and Applications Journal*, vol. 10, no. 2, pp. 93–103, 2021. |
| [10] | S. Mohsin, B. W. Salim, A. K. Mohamedsaeed, B. F. Ibrahim, and S. R. M. Zeebaree, ‘American Sign Language Recognition Based on Transfer Learning Algorithms’, *International Journal of Intelligent Systems and Applications in Engineering*, vol. 12, no. 5s, pp. 390–399, 2024. |
| [11] | “Hand landmarks detection guide | MediaPipe,” *Google Developers*. https://developers.google.com/mediapipe/solutions/vision/hand\_landmarker |
| [12] | M. Gurucharan, “Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network,” *upGrad blog*, Dec. 07, 2020. https://www.upgrad.com/blog/basic-cnn-architecture |
| [13] | “Understanding VGG16: Concepts, Architecture, and Performance,” *Datagen*. <https://datagen.tech/guides/computer-vision/vgg16/> |
| [14] | “PyTorch,” *www.pytorch.org*. <https://pytorch.org/hub/pytorch_vision_inception_v3/> |
| [15] | “Understanding ResNet50 architecture,” *OpenGenus IQ: Learn Computer Science*, Mar. 30, 2020. https://iq.opengenus.org/resnet50-architecture/ |
| [16] | “PyTorch,” *www.pytorch.org*. https://pytorch.org/hub/pytorch\_vision\_densenet/ |