Convolutional Neural Networks for American Sign Language(ASL) characters Recognition

Yuyang, Zhang   
University of FloridaGainesville, Florida  
yuyoungzhang@gmail.com  
Hang, Shu  
University of FloridaGainesville, Florida  
hangshu@ufl.edu  
Xinrun, Li  
University of FloridaGainesville, Florida  
xinrunli@ufl.edu

***Abstract—ASL recognition is one of the marquee problems in the world of machine learning. Convolutional Neural Networks show great potential for recognizing ASL, and this can be leveraged to improve the performance of ASL recognition systems. In this report a convolutional neural network was implemented to classify 9 different ASL characters. A CNN model was trained by using 8443 ASL character images. This research shows the massive potential for convolutional neural networks to be extremely accurate ASL classifiers.***

***Keywords—Machine Learning, Convolutional Neural Network (CNN), Transfer Learning with VGG16, ASL***

# Introduction

Artificial Intelligence is a significant tool that transforms learning and problem-solving approaches. It takes advantage of advanced algorithms and computational capabilities, ushering in a new era of technological applications across various domains. Machine learning forms the essential foundation of artificial intelligence, employing extensive datasets and statistical algorithms for human-like learning and adept pattern recognition. This essay focuses specifically on a representative area of machine learning — image recognition, with a particular emphasis on American Sign Language (ASL) characters.

Many machine learning algorithms have been employed in the domain of ASL character recognition, showcasing diverse approaches and outcomes. A significant study proposed the recognition of American Sign Language using Support Vector Machine (SVM) [1]. This research focused on extracting features from the dataset and applying SVM for ASL alphabet classification, demonstrating the effectiveness of this method in sign language recognition. Another investigation achieved sign language recognition rates using SVM of 80.30%, showing its potential in distinguishing between static and dynamic gestures [2]. Additionally, K-Nearest Neighbors (KNN) has been utilized for real-time sign language identification. This approach proved particularly helpful for those learning sign language, reflecting the adaptability of KNN in ASL recognition [3]. To further emphasize the breadth of these methodologies, a comprehensive study analyzed systems developed using SVM and KNN, among other techniques, providing a broader perspective on the technological advancements in sign language recognition systems [4]. While these studies present promising results, continuous refinement and exploration of advanced algorithms tailored to the intricacies of ASL characters remain meaningful for achieving higher levels in this specialized domain.

In the process of searching literature, it can be found that many remarkable results are obtained by neural networks. Each layer in a neural network identifies different features in an image, starting from basic patterns to more complex details, allowing the network to progressively understand and interpret visual data. This hierarchical processing, combined with the ability to learn from large datasets, makes neural networks exceptionally adept at recognizing and classifying images with high accuracy.

Moreover, introducing Convolutional Neural Networks (CNNs) into the neural network framework enhances its capabilities for image-related tasks. CNNs incorporate specialized convolutional layers, leveraging filters to capture intricate spatial hierarchies within visual data. This integration elevates the network's proficiency in tasks like image recognition, making it an ideal approach to implementing ASL characters Recognition.

By collecting relevant data, it can be found that CNNs have greatly promoted the development of character recognition in ASL. Another study, published in Springer, employed CNN for hand gesture recognition (HGR), focusing on both alphabets and numerals of ASL. It discussed the advantages and limitations of using CNNs, specifically highlighting architectures based on modified AlexNet and VGG16 models for classification [5]. Further, additional research described in ScienceDirect explored ASL alphabet recognition using CNN. This study tackled the challenges inherent in ASL signs, such as high interclass similarities and large intraclass variations, by employing CNN with Multiview augmentation and inference fusion [6]. Lastly, an MDPI article discussed the integration of new CNNs and deep ensemble-learning technologies in the image-recognition field, with many studies conducted to enhance ASL recognition using these advanced techniques [7].

Before implementing CNNs, we applied several morphological preprocessing operations. These operations such as dilation and erosion are significant for refining the input data. Dilation adds pixels to the character edges, emphasizing essential features for sign recognition, while erosion removes pixels from boundaries, aiding in noise reduction and clarity. Such morphological operations are pivotal in ASL recognition due to the need for precise gesture articulation. Furthermore, the application of morphological filters, contour generation, and segmentation during preprocessing contributes significantly to better feature extraction. Based on literature review and some preliminary tests, we decided to implement the task with a convolutional neural network. The dataset has 8443 300x300 labeled images, each of which shows one of A to I ASL characters, and the dataset is uniformly preprocessed before implementing the model. We employed a data partitioning method characterized by an 80-10-10 split. Finally, after several experiments, the convolutional neural network based on VGG16 was used. and an accuracy of about 87% was achieved for test set validation.

# Implementation

## Dataset

The dataset consists of a collection of ASL character images captured using camera-equipped devices. A single character from A to I is present in all images, which are shot in different backgrounds and light conditions to build a rich dataset and achieve ideal generalization abilities. In addition, color of the shooter’s skin and rotation angle of the image should be considered as well. After the summary and sorting of the images, the labeled training dataset consisted of 8443 300x300 images. The raw images were resized to 300x300 and set to RGB to create the labeled training dataset.

## Preprocessing

Pre-processing holds considerable importance in CNNs, as it not only helps in standardizing the input data, but also aids in noise reduction and feature enhancement. In our task, the process of image preprocessing involves utilizing the OpenCV library, which offers various morphological operations applicable to image data. These operations necessitate the application of a kernel, defining the pixel neighborhood considered during each operation. Through experimentation with the specific dataset, it was determined that a 4x4 kernel matrix yielded optimal results for the designated operations.

In our implementation, we employed two key morphological operations to enhance the dataset. First, the Median Blur operation is applied, which calculates a new pixel value for each point by averaging its neighboring pixels. It effectively reduces noise and smoothens the background, minimizing unnecessary details. This proves particularly valuable for our task, where some extraneous background elements are present in the images. Following the Median Blur, we executed the MORPH\_OPEN operation. The operation serves to refine characters by combining with the erosion operation. It contributes to enhancing the intricate details within the dataset. To further optimize the data, we employed the resize function, reducing each image to a compact 210x210 size. The choice of the INTER\_AREA parameter during the resizing process facilitated the necessary decimation. The dataset underwent division into a training set, a validation set, and a test set, maintaining a ratio of 80/10/10. Subsequently, pixel values were normalized by scaling them with a factor of 255. This normalization process ensured that pixel values were appropriately constrained within the standardized range of 0 to 1.

## Convolutional Neural Network (CNN)

By relevant data collected previously, it can be found that CNNs work well in image processing tasks due to their ability to efficiently extract and learn key features from visual data while maintaining the spatial relationships between pixels.

After choosing CNNs, we further utilize transfer learning to realize our recognition task. This decision was driven primarily by the challenges encountered with custom CNNs. Firstly, a custom CNN, designed from scratch, posed significant memory overheads, particularly when fitting large-sized training datasets. This often led to kernel crashes due to excessive memory usage. Secondly, tuning a custom CNN is a complex and time-consuming process, often yielding unsatisfactory performance due to the intricacy of hyperparameter adjustments. Transfer learning allows us to utilize a pre-established and robust architecture that significantly reduces the memory burden and the complexity of hyperparameter tuning.

For the implementation, we choose the VGG16 model, a proven architecture in the field of image recognition [8]. Our reasons for choosing VGG16 are explained in detail in the experiment section.

The VGG16 model includes multiple Conv2D layers. These layers are designed to perform convolution operations, which involve filtering the input images with learned kernels to extract various features. The depth of these layers in VGG16 is one of the key reasons for its high performance in image-related tasks. Each convolutional layer in VGG16 uses kernels to scan through the input image and create feature maps. These feature maps are essentially collections of filters applied to the image. This process allows the model to identify patterns and textures in the input images.

ReLU (Rectified Linear Unit) activation functions are used in the VGG16 layers. ReLU plays a meaningful role in the model because it can introduce non-linearity to the model, enabling it to learn complex patterns. A significant advantage of ReLU is its ability to mitigate the vanishing gradient problem, which is a common issue in training deep neural networks [9]. For ASL character recognition, it's vital that the model can learn and represent complex features from the images, including the shape and position of fingers and palms. ReLU activation helps in maintaining the strength of the features as it passes through the layers of the network.

Moreover, VGG16 incorporates MaxPooling2D layers. These layers are crucial for reducing the spatial dimensions of the extracted feature maps. The max pooling process in our model works by scanning a small area over the feature map and picking out the highest value from each section, which helps in reducing the computational load and focuses the model on the most important features.

Following the ideas above, we effectively utilized the strengths of the VGG16 model, reducing the need for extensive training and computational resources, and refining the model to suit our specific recognition task.

## Model Fitting and Interpretation

Once the model, incorporating the VGG16 layers, was configured, we proceeded to train it using our preprocessed training dataset. The model underwent 200 epochs, and we use early stopping and callback here to stop the model early and save the best model.During each epoch, the model's performance was monitored, and its progress was visually displayed, offering insights into the learning process in real time. After completing the training, we evaluated the model's effectiveness using our test dataset. This evaluation provided a series of scores for each epoch, illustrating the model's learning curve. To further interpret the model's performance, we plotted a graph displaying the cross-entropy loss against the number of epochs. This graph can help us visualize the model's learning curve and understand how effectively it minimized the loss over time, refining its ability to recognize the characters.

# Experiments

In our implementation, the training data is loaded first. Then we undertook a series of preprocessing and data splitting steps to prepare our training dataset for the CNN.



Figure 1: ASL character image before(L) and after(R) preprocessing

We started by reshaping each image in our training dataset to a size of 300x300 pixels with 3 color channels. To enhance the image quality, we applied a median blur using OpenCV's medianBlur function, which helped reduce noise. We then performed morphological opening with a 4x4 kernel to eliminate small, irrelevant objects, thereby cleaning up the image background. Finally, each image was resized to 210x210 pixels for uniformity.

In our project, we divided the preprocessed images into training, validation, and test sets using an 80-10-10 split, employing Scikit-learn's train\_test\_split function for an even distribution while maintaining class balance and ensuring reproducibility with a fixed random\_state. Subsequently, we converted all datasets into TensorFlow constants of type tf.float16 to optimize memory usage and normalized pixel values from 0-255 to 0-1, enhancing the network's feature sensitivity and training efficiency.

In step six of our project, we meticulously crafted the architecture of our CNN model, integrating the VGG16 model through transfer learning. The rationale behind incorporating VGG16 was to harness its pre-existing knowledge of image features, proven through our experiments to enhance accuracy. The decision to employ VGG16 was substantiated by experimental results, where it outperformed other models available in Keras with pre-trained ImageNet weights in terms of accuracy. This finding aligns with similar studies and applications found on platforms like Kaggle, where VGG16 has been successfully applied to ASL recognition tasks [10]. We configured VGG16 to accept input tensors matching our image shape and set include\_top to False, thereby excluding its fully connected layers, a decision validated by improved performance in our tests.

In preserving the integrity of VGG16's learned features, we set its trainable attribute to False, ensuring the stability of its weights during our training. Beyond the VGG16 base, our model included a BatchNormalization layer, essential for normalizing inputs to the subsequent layers and enhancing network stability.

In addition, we incorporated two dropout layers in our model. The first Dropout layer, with a rate of 0.5, followed the VGG16 model output. This layer played a pivotal role in reducing overfitting by randomly setting a portion of input units to zero during training, thus ensuring the model did not rely too heavily on any single feature. The second Dropout layer, with a rate of 0.7, was placed after a Dense layer of 256 units with ReLU activation. Notice that the rate of this layer should be modified to 0.7 to avoid overfitting. This additional dropout layer further augmented the model's generalization capabilities, a strategy supported by our iterative testing which indicated a reduction in overfitting without significantly compromising model performance.

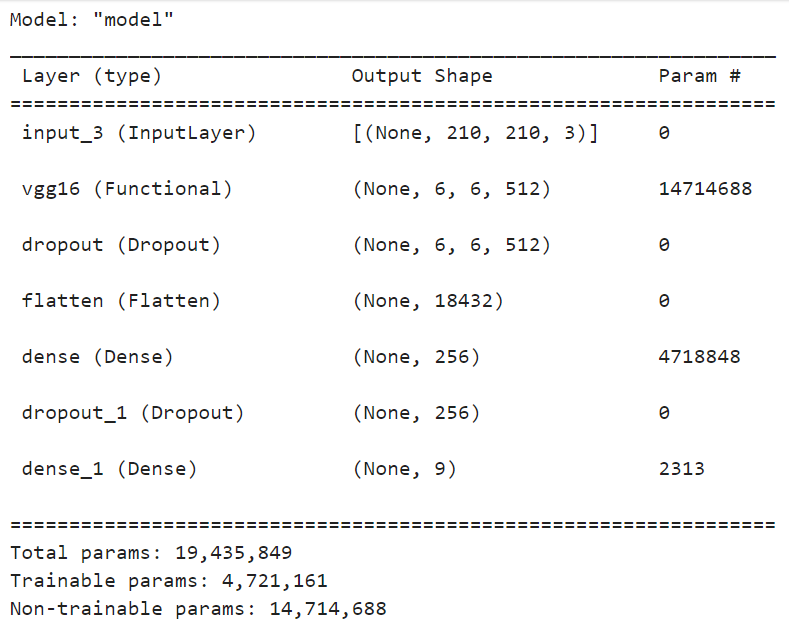


Figure 2 CNN’s Architecture

The model's architecture was rounded off with a final Output layer of 9 outputs, corresponding to the classes in our ASL dataset, utilizing a softmax activation function to output a probability distribution across these classes. The model was compiled with the Nadam optimizer and sparse\_categorical\_crossentropy as the loss function, chosen for its suitability for multi-class classification tasks, and accuracy as a performance metric.

Through this carefully considered architecture, incorporating two strategically placed dropout layers, we aimed to balance the model's capacity to learn complex features with its ability to generalize effectively to new data.

In step seven of our project, we assessed our CNN model's performance by defining an “Evaluate” function, which visualized the model's training and validation accuracy and loss across epochs, providing key performance metrics.

We fine-tuned the learning rate of our CNN model using grid search. Utilizing Scikit-learn's GridSearchCV with a range of learning rates [0.0001, 0.00005, 0.00001], we integrated our Keras model using KerasClassifier. The grid search, involving 3-fold cross-validation over 200 epochs, identified the optimal learning rate by evaluating the model's performance for each rate. Upon completion, the best learning rate was determined from the grid search results, significantly enhancing our model's training efficiency.

Next, we trained our CNN model using the optimal learning rate of 0.00001, identified from the grid search, and set in our define\_model function with the VGG16 architecture and custom layers. We incorporated Early Stopping to halt training if there was no improvement in validation loss for 10 epochs, and used the Nadam optimizer for efficient learning. Data augmentation with ImageDataGenerator, including random rotations, enhanced the model's robustness and improved accuracy by 1.5 to 2 percent, despite increasing memory usage and training time. The model, trained for 200 epochs with batch size 32, showed a steady improvement in accuracy and decrease in loss, highlighting the effectiveness of our training approach.

Then, we evaluated the performance of our CNN model, trained with VGG16 transfer learning, on both the test set and a 10% subset of the training data. We loaded our saved model 'model.h5', ensuring its architecture matched our expectations, and then used the Evaluate\_performance function to assess accuracy, precision, recall, and F1-score for each class, as well as to generate a confusion matrix. This analysis revealed the model's overall accuracy to be approximately 86.5% on the subset, providing insights into its strengths and potential areas for improvement.

At last, we trained the final model using the entire training dataset and the same model structure, adjusting the epochs to 10 less than the final epoch number used for the 80% training set. The final model, trained for 200 epochs with a learning rate of 0.00001, demonstrated an accuracy of approximately 92.69% and a decreasing loss trend, indicating effective adaptation to the ASL dataset despite some overfitting.

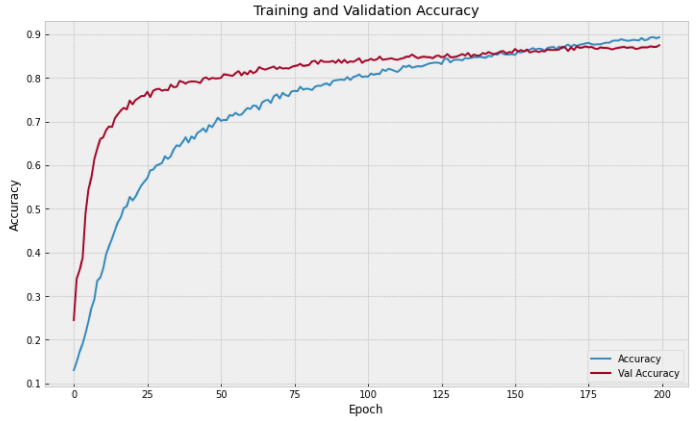


Figure 3 Training and Validation Accuracy

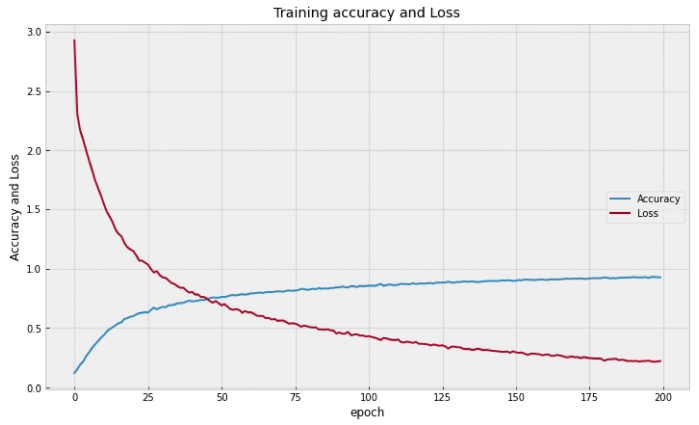


Figure 4 Training Accuracy and Loss of the trained CNN

# Conclusions

Upon concluding the research presented in this paper, we have demonstrated that CNNs stand as a highly effective method for image classification tasks in machine learning. The CNN, with its intricate convolutional layers, proved superior in accuracy compared to other methods we tested. These layers are particularly adept at recognizing complex patterns, showcasing their immense potential in image processing and computer vision applications.

Our implementation of CNN, especially with the integration of transfer learning using the VGG16 model, achieved a classification accuracy of over 86%. This accomplishment underscores the capability of CNNs in handling intricate image recognition tasks. We believe that future enhancements in the dataset's robustness, coupled with extended training durations and more powerful computing resources, have the potential to further elevate the model's performance.

##### References

1. Jain, Vanita et al. “American Sign Language recognition using Support Vector Machine and Convolutional Neural Network.” International Journal of Information Technology, Volume 13 (3) – Feb 25, 2021.
2. “American Sign Language Recognition Using Leap Motion Controller with Support Vector Machine and Deep Neural Network.” MDPI.
3. “Real-Time Sign Language Identification using KNN: A Machine Learning Approach.” IEEE Xplore.
4. “A comprehensive study on sign languages recognition systems using SVM, KNN, CNN, and ANN.” ACM Digital Library. October 2018.
5. “CNN based feature extraction and classification for sign language.” Springer. Published: 18 September 2020.
6. “American Sign Language alphabet recognition using Convolutional Neural Networks.” ScienceDirect. November 2018.
7. “Ensemble Learning of Multiple Deep CNNs Using Accuracy-Based Approaches for ASL Recognition.” MDPI. 2022.
8. Simonyan, K.; Zisserman, A. “Very Deep Convolutional Networks for Large-Scale Image Recognition.” 2014.
9. He, K.; Zhang, X.; Ren, S.; Sun, J. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.” Proceedings of the IEEE International Conference on Computer Vision. 2015.

[10] ASL Alphabet Classification with VGG16 on Kaggle