ASL Character Classification

Akshat Kothiyal
Computer and Information Sciences(CISE)
University of Florida
kothiyalakshat@ufl.edu [UFID:53292487]

Priyanshu Prasad
CISE
University of Florida
pprasad@ufl.edu [UFID:73804925]

Shiva Shankar Katika
CISE
University of Florida
s.katika@ufl.edu [UFID:94165817]

***Abstract*—The project presents an ambitious venture in the realm of assistive technology, focusing on the development of a sophisticated machine learning system designed to classify American Sign Language (ASL) characters. This endeavor aims to bridge the communication gap between the deaf and hearing communities by leveraging advanced computational techniques. The project involves capturing ASL gestures through image inputs and employing deep learning algorithms to accurately interpret these signs, using a custom dataset that was used in our project. The dataset, specifically created for ASL gestures, encompasses a diverse range of signs in different orientations and lightings to ensure the robustness of the model. Central to this project is the application of the VGG16(Visual Geometry Group 16) [1] convolutional neural network. By integrating the VGG16 architecture as a cutting-edge deep learning system, known for its effectiveness in image recognition tasks, the system captures and interprets ASL gestures through image inputs.The system's efficacy is gauged through several parameters, with a primary focus on accuracy. Accuracy is critical in ensuring that the translations of ASL characters are correct, thus making reliable communication possible.**

—VGG16(Visual Geometry Group 16), American Sign Language (ASL), convolutional neural network, deep learning

I. Introduction

Sign language is the primary form of communication used by people who have impaired hearing and speech. While it is the primary form of communication for these people it is hard for other people to understand them. So there should be an efficient form of system that can interpret these signs. Many researchers in the field have leveraged the machine learning and deep learning approaches to detect the hand gestures.

Kshijit bantupalli and Yang Xie in their paper proposed a model that is the combination of Inception-based CNN and LSTM-based RNN that detects the hand gestures. They used a CNN (Convolutional Neural Network) model called Inception to extract spatial features from the video stream. Then, they extracted temporal features from the video sequences using an RNN (Recurrent Neural Network) model called an LSTM (Long Short-Term Memory). Their model performed poorly when there are faces in the video streams and if there are different skin tones. [2]

Nikhil Kasukurthi, Brij Rokad et.al in their paper used a squeeze net architecture and trained the model for hand gestures. They used a squeeze net architecture to run the model in the mobile devices. They got a validation accuracy of 83.29%. [3]

Sharmila Konwar et.al, in their paper used a set of operations to get a morphological image of a hand to use them for hand gesture recognition. They used a HSV color space for skin detection and they used a specific set of parameters namely hue , saturation and luminance to define the boundary of skin pixels and they used a edge detection algorithm called canny edge detection algorithm to detect the edges of the hand gesture. The edge detected image from this paper can be used by deep learning models to efficiently classify the images. [4]

Sakshi Sharma et,al, in their paper compared the accuracies of vgg-11 and vgg- 16 for Indian sign language detection and also proposed a G-CNN model for hand gesture classification and were able to achieve a accuracy of 94.83%.[5]

Jaya Prakash Sahoo et.al in their paper developed a system that consists of fine tuned Alex net and Vgg-16 and a score fusion is performed on the outputs of the neural networks and a decision was made using the fused score. Using the LOO CV test on the MU dataset and the HUST-ASL dataset, the suggested method performs at 90.26 and 56.18 mean accuracy, respectively. Comparably, the suggested technique's performance on the standard CV test yields mean accuracy values of 64.55% for both datasets and 98.14% for the former.[6]

The decision to employ the VGG16 architecture in this project is underpinned by several compelling considerations. Firstly, the inherent complexities of American Sign Language (ASL), characterized by nuanced hand gestures and movements, necessitate a deep learning model capable of capturing intricate patterns. VGG16's deep architecture aligns well with these requirements, providing a robust framework for recognizing the subtle intricacies embedded in ASL expressions. Secondly, the model's well-documented success in diverse image classification tasks underscores its potential for accurately discerning ASL characters, instilling confidence in its efficacy for our specific application.

The organization of the paper is as follows; Section 2 highlights the model used and its Implementation. The model experiments are outlined in Section 3. The results obtained and conclusions are presented and discussed in Section 4.

II. Implementation

A. Overview

Our project implements a machine learning system for classifying American Sign Language (ASL) characters. The model is based on the VGG16 architecture, adapted and fine-tuned for ASL character recognition.

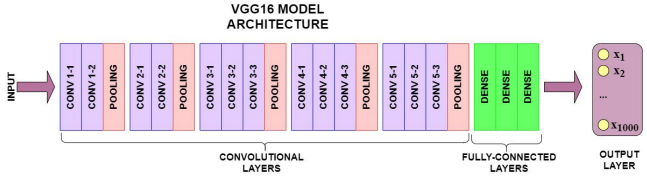

Fig. 1. VGG-16 (CNN Architecture)

B. Data Set

The data set for this project has been collected as part of our course, comprising of 8443 images of ASL characters, with each image labeled according to the represented character of 9 classes of American sign language.

C. VGG-16 (CNN architecture)

VGG16 is a convolutional neural network (CNN) architecture proposed by Karen Simonyan and Andrew Zisserman in 2014. It achieved state-of-the-art performance on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in the same year. The VGG 16 consists of 13 convolution layers , 5 max pooling layers and 3 fully connected layers. The convolution layer consists of small 3x3 filter size and increasing number of filters (32 to 512). There are 5 max pooling layers applied after each group of convolutional layers to reduce the spatial dimensions and 3 fully connected layers with Dense layers with 4096 units each, followed by a final 1000 unit softmax layer for classification.[1]

D. Image Preprocessing

The data preparation involves the following steps:

1) Loading Data: : The images (‘data_train.npy‘) and labels (‘labels_train.npy‘) are loaded using NumPy.

2) Splitting Dataset: :he data is divided into training, validation, and test sets using ‘train_test_split‘ from Scikit-Learn.

3) Reshaping: : The input data is reshaped to have dimensions (300, 300, 3) to match the input shape expected by the VGG16 model.

4) Data Transposition: :Each image in the dataset is transposed to align the image data correctly.

5) Normalization: : The pixel values of the images are normalized to the range [0, 1] by dividing them by 255.0.

6) Label Categorization: : The labels are converted into categorical format using Keras’ ‘to_categorical‘, facilitating compatibility with the model’s output layer.

E. Image batch generator

The code incorporates an essential component known as the ”image_batch_generator” function, designed to facilitate the training of deep learning models. This custom generator is instrumental in efficiently handling large datasets by dynamically creating batches of images and their corresponding labels during the training process. Operating in an infinite loop, the generator ensures a continuous flow of diverse training samples. For each iteration, the generator extracts a batch of data and reshapes it to the specified image dimensions, normalizing pixel values to a standardized range.

F. Model Architecture

We adapted the VGG16 model for our ASL classification task:

1) Base Model: : VGG16, pre-trained on ImageNet, without its top layer, is used as the base.

2) Layer Customization: : The last five layers of VGG16 are unfrozen to allow fine-tuning.

3) Adding Custom Layers: : A Flatten layer follows the base model, then a Dense layer with 1024 neurons, applying ReLU activation and L2 regularization. A Dropout layer with a rate of 0.5 is included to prevent overfitting.

4) Output Layer: : A final Dense layer with 9 neurons (corresponding to the number of ASL classes) uses a softmax activation function for multi-class classification.

The code incorporates an essential component known as the ”image_batch_generator” function, designed to facilitate the training of deep learning models. This custom generator is instrumental in efficiently handling large datasets by dynamically creating batches of images and their corresponding labels during the training process. Operating in an infinite loop, the generator ensures a continuous flow of diverse training samples. For each iteration, the generator extracts a batch of data and reshapes it to the specified image dimensions, normalizing pixel values to a standardized range.

G. Compilation

The model is compiled with the RMSprop optimizer, a learning rate of 0.0001, and using categorical cross-entropy as the loss function.

H. Training

The model is trained using the following approach:

1) Batch Generator: : An image batch generator function is implemented to feed data into the model in batches of size 32.

2) Early Stopping: : To prevent overfitting, early stopping is employed, monitoring the validation loss and stopping the training when it ceases to decrease.

3) Training Process: : The model is trained for up to 100 epochs, but typically stops earlier due to early stopping.

I. Evaluation

Post-training, the model’s performance is evaluated on the test set. We use accuracy as the metric, calculated by comparing the model’s predictions against the true labels of the test data.

III. Experiments

A. Experiment Setup

In this section, we explore the effects of various hyperparameters on the performance of our ASL hand sign classification model. Our experimentation focused on fine-tuning the hyperparameters of a VGG16-based model for ASL hand

sign classification. We conducted a series of experiments to understand how different settings impact the model's accuracy and generalization capabilities.

We used HiperGator, the University of Florida's super computer to train our model. We used the following architecture to train our model.

B. Learning Rate

Lower Learning Rate (e.g., 0.0001): Resulted in slower convergence, but often led to better generalization. The model took more epochs to reach optimal performance, yet it was less likely to overfit.

Higher Learning Rate (e.g., 0.01): Led to faster convergence but often caused instability in training. In some cases, the model quickly reached a suboptimal solution, indicating overfitting.

C. Batch Size

Chosen Batch Size (32): This size was found to be the optimal balance between memory usage and model performance. Larger batch sizes, such as 64, led to memory overload issues both on a personal computer and on the university's supercomputer, HiperGator. A batch size of 32 provided sufficient data for each training iteration without exceeding memory limits.

D. Dropout Rate

Settled on Dropout Rate (0.5): A higher dropout rate was crucial to mitigate overfitting. Given the complexity of the model and the richness of the dataset, a dropout rate of 0.5 provided a strong regularization effect. This prevented the model from becoming overly reliant on any particular neuron within a layer, thus enhancing its generalization capabilities on unseen data.

E. Number of Epochs and Early Stopping

Early Stopping with 100 Epochs: We implemented early stopping to prevent overfitting and to optimize training time. The model typically stopped training around 10-15 epochs, indicating that the model quickly reached a plateau in performance. This approach ensured that the model did not continue to train unnecessarily, which could lead to overfitting and wasted computational resources.

F. L2 Regularization Coefficient

Lower Coefficient (e.g., 0.001): Resulted in minimal regularization, which was not always enough to combat overfitting in our model.

Higher Coefficient (e.g., 0.01): Introduced significant regularization, which helped in reducing overfitting but occasionally led to underfitting.

G. Unfreezing Layers in VGG16

Unfreezing the Last 5 Layers: The decision to unfreeze the last five layers of the VGG16 model was based on the need to fine-tune more advanced features specific to ASL signs. While the initial layers of VGG16 capture generic features like edges and textures, the latter layers are more adept at learning complex patterns. By unfreezing these layers, we allowed the model to adapt these complex feature detectors to the specific nuances of ASL signs, thereby improving its overall predictive performance.

H. Activation Function

We used the Rectified Linear Unit (ReLU) for its efficiency in introducing non-linearity, crucial for image classification. ReLU helps overcome the vanishing gradient problem, enabling the model to learn complex patterns. Future experiments could explore other functions like Leaky ReLU or ELU to address any limitations of ReLU, such as the issue of dead neurons.

I. Activation Function

RMSprop was chosen for its effectiveness in dynamic learning rate adjustment and handling noisy gradients. This optimizer significantly influenced the speed and stability of the model's training process. Other common optimizers like Stochastic Gradient Descent (SGD) and Adam were considered. However, RMSprop provided the most accuracy.

J. Number of Neurons in Dense Layer

The dense layer was configured with 1024 neurons, balancing the model's ability to capture intricate patterns and its overall complexity. This number of neurons provided sufficient capacity for the model to differentiate between various hand signs while managing the risk of overfitting. Adjusting the number of neurons could be an area for further experimentation, especially with changes in data availability and complexity.

CONCLUSION

This project successfully demonstrates the feasibility and effectiveness of using a modified VGG16 convolutional neural network for the classification of American Sign Language characters.The modified VGG16 architecture proved to be highly adaptable for the task of ASL character recognition. The decision to unfreeze the last five layers allowed the model to fine-tune pre-trained features specific to our dataset, enhancing its accuracy. During training, the model achieved an impressive accuracy of 99%, showcasing its ability to correctly predict ASL gestures on the training dataset. This high training accuracy suggests that the model effectively learned the intricacies and patterns within the provided ASL data.

Upon evaluation on the validation set, the model exhibited a commendable accuracy of 92%. This validation accuracy provides confidence in the model's generalization capabilities, indicating its proficiency in recognizing ASL gestures on previously unseen data. The validation results are crucial as they validate the model's ability to perform well beyond the training set, reflecting its potential for real-world applications.Implementing an early stopping mechanism was crucial for efficient training. It prevented overtraining and helped in achieving a good balance between the training and validation accuracy.

Furthermore, the model demonstrated robust performance on the test set, achieving an accuracy of 93%. This independent evaluation on a distinct subset of data reaffirms the model's reliability and underscores its capacity to generalize to diverse ASL gestures. The project also shed light on practical challenges, such as memory limitations encountered with larger batch sizes. This highlighted the importance of considering computational resources in model design and experimentation.While the results are promising, there is room for further improvement. Future work could explore more advanced techniques in data augmentation, experiment with different architectures, or apply transfer learning from other image recognition tasks.

REFERENCES

[1] S. Liu and W. Deng, "Very deep convolutional neural network based image classification using small training sample size," 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR), Kuala Lumpur, Malaysia, 2015, pp. 730-734, doi: 10.1109/ACPR.2015.7486599.

[2] Bantupalli, Kshitij and Xie, Ying. (2018). American Sign Language Recognition using Deep Learning and Computer Vision. 4896-4899. 10.1109/BigData.2018.8622141.

[3] Kasukurthi, Nikhil and Rokad, Brij and Bidani, Shiv and Dennisan, Dr. (2019). American Sign Language Alphabet Recognition using Deep Learning.

[4] Konwar, A. and Borah, B. and Tuithung, C.T.. (2014). An American Sign Language detection system using HSV color model and edge detection. International Conference on Communication and Signal Processing, ICCSP 2014 - Proceedings. 743-747. 10.1109/ICCSP.2014.6949942.

[5] C J, Sruthi and Lijiya A. "Signet: A Deep Learning based Indian Sign Language Recognition System." 2019 International Conference on Communication and Signal Processing (ICCSP) (2019): 0596-0600.

[6] Sahoo, Jaya and Ari, Samit and Patra, Sarat. (2021). A user independent hand gesture recognition system using deep CNN feature fusion and machine learning technique. 10.1016/B978-0-12-822133-4.00011-6.

[7] https://www.tensorflow.org/

[8] https://developer.nvidia.com/cuda-toolkit