AMERICAN SIGN LANGUAGE ALPHABET RECOGNITION AND TRANSLATION

Nenad Panić*

University Singidunum,
Belgrade, Serbia

Abstract:

This paper presents a study on American Sign Language (ASL) alphabet recognition and translation. ASL is a complex language used by the Deaf community for communication. With the increasing dependency on technology in our daily lives, and with the increasing adoption of working from home, the successful inclusion of these communities in video calls and meetings is of great importance. With great advancements in Artificial Intelligence over the past years, it has now become possible to build, train and use proven to be powerful image-processing convolutional neural networks (CNN) for successful sign language recognition and translation into English text. The models have been trained and tested on a University of Exeter-derived dataset and have achieved high accuracy in this task. We have compared the results of several pre-trained models using transfer learning, as well as our own CNN. Our study shows great possibilities for improving communication between the Deaf and hearing communities.

Keywords:

Convolutional Neural Network, Image processing, Transfer learning, American Sign Language.

INTRODUCTION

Expanding on the assistive technologies present today is more important than ever before. Recent world events have increased our dependency on technology and have pushed many into a new way of working from home. To give every community, and every person an equal chance of successfully navigating this new way of living, assistive technology advancements must continue.

We have embarked in this study upon a challenge of recognizing and translating American Sign Language (ASL), which is a natural language that is utilized by individuals who are deaf or hard of hearing in the United States and other parts of the globe. Despite being a complex and expressive language, the fact that not everyone is proficient in ASL presents unique communication and interaction challenges with those who are hearing. Recent advances in Machine Learning and the proven ability of Convolutional Neural Networks to classify images more and more accurately have made it possible to develop systems that can recognize and translate American Sign Language in real-time.

Correspondence:
Nenad Panić

e-mail:
nenad.panic.22@singimail.rs

DOI: 10.15308/Sinteza-2023-312-319

Sign language in general relies of course on static signs but as well on movement and hand position. Successful translation and recognition of ASL consists of training the model on a large dataset of hand signs and gestures. In this study, we will be translating only static signs. The reason behind this is that we have made use of a dataset that consists only of images, therefore the study conducted focuses on static signs of ASL alphabet letters. The focus is on successfully classifying and translating 24 letters of the alphabet A-I, K-Y. Letters J and Z have been excluded from the classification as movement is required for their representation [1]. Letters P & K, as well as Q & G, use the same sign but in a different position, but as the distinction can be made in the dataset for these sets of letters, the images that represent these letters have been placed in their respectable classifications.

CNNs are a class of deep learning algorithms that can be applied to a wide range of image classification, object recognition as well as facial recognition tasks. They can either be created from the beginning or pre-trained CNNs can be used that have been created already by large organizations.

This concept of using someone else's neural network is called "Transfer learning" [2]. The main reason behind transfer learning is that the actual training of convolutional neural networks requires a lot of resources and a lot of data. Therefore, organizations with access to greater computing power and data build and train complex models that otherwise could not have been able to be made by any individual and let other people use their models for their purposes. Transfer learning models are made to be as general as possible within image classification in this case, so that they can be applied to any problem. The borrowed models can be tested on the dataset to which our problem is related and if needed certain parts and weights of the borrowed model can be adjusted to fit our dataset better. It is important to limit the amount of re-training allowed so we don't completely diminish the current weightings created by extensive training on large datasets, otherwise, transfer learning models will lose their value. It is also possible to add extra layers of our own on top of the pre-trained model. In this study, we have made use of both approaches.

We have created a CNN from the beginning and have used pre-trained models such as ResNet50, MobileNetV2, and VGG19. ResNet50 is a 50-layer convolutional neural network consisting of 48 convolutional layers, one Max Pool layer, and one average pool layer [3]. The MobileNet-v2 is a convolutional neural network that has 53 layers and has undergone extensive training using a vast ImageNet database. This pre-trained network is capable of categorizing images into 1000 distinct object categories, including animals, pencils, keyboards, mice, and more [4]. VGG19 is a convolutional neural network that has 19 layers and has also been trained on the ImageNet database, capable of classifying images into 1000 categories [5]. We have added several extra Dropout, Dense, and Batch Normalization layers to all the transfer learning models mentioned above. In this paper, we will examine the architecture of the models used, training and testing principles, and performance. We will also discuss current limitations and future directions for improvement.

2. DATASET

The dataset itself has been derived from the University of Exeter's [6] website. It consists of 55,000 images. The images have been preprocessed into a 160x160 format with 3 color channels and have been split into two subsets. Training and test. Inside every subset, there is a folder for every classification of images. The class names have been transformed from categorical to numerical. A few data samples with their respective categorical classifications can be seen in Figure 1.

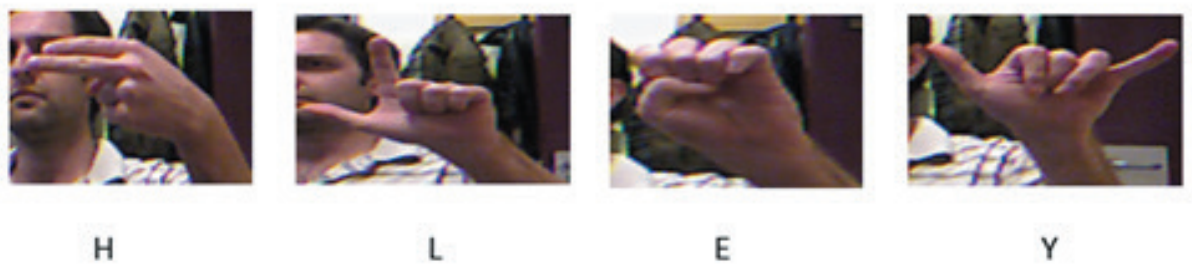

Figure 1 – Sample images from the dataset with their classification included.

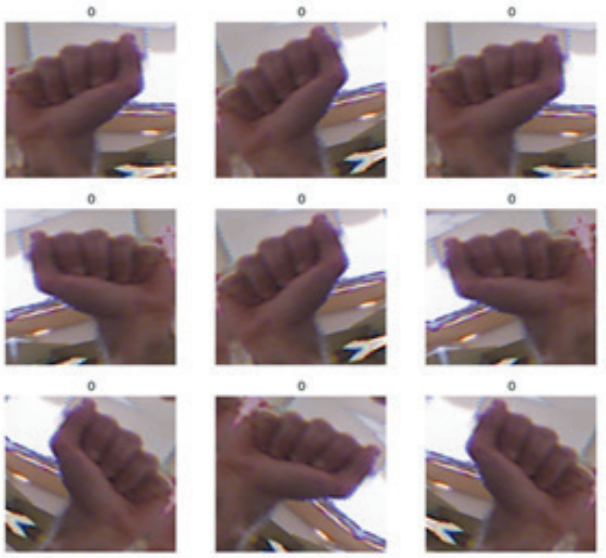

Figure 2 – Data augmentation example

Since the dataset is not very large and diversified, to decrease overfitting as much as possible we have used Data augmentation techniques such as Random Zoom, Random Flip, and Random Rotation. An example of the result of this technique on a single image can be seen in Figure 2.

3. APPLIED METHODS AND ARCHITECTURES OF CONVOLUTIONAL NEURAL NETWORKS.

The CNNs can have an array of different kinds of layers within them. We will explain shortly what each of the layers we have used in the study are used for:

- The fundamental building block of a CNN is the convolutional layer. It contains a set of filters and parameters which are learned through the process of training. The filters convolve over the images and the dot product of the filter and the underlying pixel values is calculated at every position. After this operation has been done for the entire image, the feature map or input for the next layer has been formed [7];
- Pooling layers are usually placed right after convolutional layers since they provide an approach to down-sampling feature maps. Therefore, they reduce the number of parameters a network needs to learn, and the amount of computation performed in the network. The pooling layer summarizes the features present in a certain region of the feature map that has been generated by the convolutional layer. This makes the model more robust to variations in the position of the features in a given image. There are two types of pooling techniques. Max pooling involves identifying the maximum value element within a specific area of the feature map that is encompassed by the filter. The output is then a feature map that contains all the most prominent features of the previous feature map. There is one other technique of pooling, which is Average pooling. It calculates the average value of the elements present in the region of the feature map covered by the filter and creates a new feature map with the averages from each region [7];
- The dropout layer is responsible for randomly selecting a given number of neurons to ignore during training. This is most frequently done to reduce the effects of over-fitting;
- The flattening layer converts the data into a 1-dimensional array for input into the next layer; and
- A dense layer, as with any other neural network, establishes a deep connection with the preceding layer by interconnecting the neurons of the layer with each neuron of the preceding layer.

3.1. CUSTOM CONVOLUTIONAL NEURAL NETWORK

We have implemented our neural network using TensorFlow's Keras API. The network itself consists of three convolutional layers, three max-pooling layers, two dropout layers, and two dense layers.

The model takes the input images and outputs a classification prediction for one of the 24 classes. Here is a brief description of the layers in the model:

1. Data augmentation: The first layer of the network is a data augmentation layer, which applies the previously mentioned augmentation techniques to images to increase the size of the training dataset;
2. Rescaling: This layer rescales the input pixel values from a range of [0, 255] to [0, 1];
3. Conv2D: This layer performs convolution on the input image using a set of learnable filters. The layer has 16 filters, a filter size of 3x3, and a ReLU activation function. Padding has been used to preserve the images in 160x160 format;
4. MaxPooling2D: This layer downsamples the feature maps produced by the convolutional layer;
5. Conv2D: This layer performs convolution on the output of the previous MaxPooling2D layer. This layer has 64 filters, a filter size of 3x3, and a ReLU activation function. We have again used padding to preserve the image dimensions;
6. MaxPooling2D: Just like in the previous max pooling layer, this layer downsamples the feature maps of the convolutional layer and produces a feature map of the strongest features;
7. Dropout: This layer randomly drops out some of the features during training to reduce overfitting. We have found in this case that a figure of 50% works best;
8. Conv2D: This is the last convolutional layer in the network. It has 64 filters, a size of 3x3, and a ReLU activation function. No padding has been used in this layer;
9. MaxPooling2D: This is the last max pooling layer, producing the final feature map that will be used for classification;
10. Dropout: We have again decided to include an aggressive dropout layer with a 60% dropout rate due to the problem of overfitting;
11. Flatten: This layer flattens the output of the previous MaxPooling2D layer to create a 1-Dimensional vector.
12. Dense: This fully connected layer has 128 units and a ReLU activation function. It applies L2 regularization with a coefficient of 0.01 to the activity of the layer; and
13. Dense: This fully connected layer has 24 units and no activation function. It produces the final classification output.

The optimizer used during training is RMSProp. The learning rate used in this model is 2e-5, a small value has been used to ensure that the model parameters are updated slowly during training to improve model convergence. Callback has also been used to train this model with a monitor for validation loss. The model has been trained in 50 epochs with a batch size of 128.

3.2. PRE-TRAINED MODELS

The pre-trained models used in this study are VGG19, ResNet50, and MobileNetV2. Let us go over our implementations of each of them. A general approach that we have found to produce the best results is to only include the base of the pre-trained model and our own dense and classification layers.

3.2.1. VGG19

The VGG19 implementation in this study includes the base model as well as 8 additional layers.

The first additional layer after the base model is a flattening layer, that creates a 1-dimensional array of the base model's output. A Dense layer follows with 512 units and a ReLU activation function. We have then used a dropout layer with a rate of 40%, a batch normalization layer, another Dense layer with 512 units, and a ReLU activation function followed by a dropout layer with a rate of 50% and a batch normalization layer one more time. The final layer is a dense layer with 25 units, using a SoftMax activation function and an L2 regularizer with a coefficient of 0.01. The model has been trained with the use of an RMSProp optimizer with a learning rate of 1e-5 with a Sparse Categorical Cross entropy loss function.

3.2.2. Mobile Net V2

The Mobile Net architecture has been used without its top layers, and the trainable parameter has also been set to False, which freezes the weights of the pre-trained layers in the Mobile Net to avoid overfitting. We have also used the techniques of data augmentation as well as preprocessed the input using the pre-defined Mobile Net pre-processing function which scales the pixel values from [0, 255] to [-1, 1]. Just like in the case with VGG19, after the base model output we have implemented a flat-

tening layer followed by a dense layer with 512 units and a ReLU activation function. Followed by a dropout layer with a rate of 50%, a dense layer with the same configuration, another aggressive dropout layer with a rate of 70%, followed by the last and dense layer with 24 units, SoftMax activation, and L2 regularizer with a coefficient of 0.01. The model has been trained using the RMSProp optimizer with a learning rate of 1e-5 with a validation loss callback function. It has been trained in 25 epochs with a batch size of 128.

3.2.3. ResNet 50

For the implementation of the ResNet architecture, we have set the trainable attribute to True, allowing the whole model to adapt during training. As well as for previous models the first layers are for data augmentation and scaling. Scaling has been done the same as for Mobile Net with pixel values ranging from -1 to 1. Then the input will go through the base ResNet model. The additional 8 layers are similar to the previous models, with the first being the flattening layer, followed by a 512-unit dense layer with the ReLU activation function, a dropout layer with a rate of 50%, batch normalization, another identical dense layer, dropout layer with a rate of 70%, batch normalization layer and the last layer being the dense layer with 24 units, SoftMax activation and L2 regularizer with the coefficient of 0.01. This model has been trained with the Adam optimizer with a learning rate of 1e-5 with a Sparse categorical cross-entropy loss function measuring accuracy, and a callback validation loss function.

4. RESULTS

We have measured the training and test accuracy of our models. We have as well measured the training and validation loss. Here we will discuss the results obtained from each of the models used and compare them among the models. A figure showing the training and validation curves will be shown for every model.

Let us begin by showing the results of our neural network. As can be seen in Figure 3, the training accuracy is reasonably high at 89.53% and training loss has become very low by the 50th epoch. We can conclude that as the number of epochs rose the training accuracy and training loss have become better and better. Unfortunately, the opposite can be said for validation loss and validation accuracy. The model has achieved a validation accuracy of a mere 67.03%. During the first 20 epochs, the validation accuracy and validation loss have exponentially gotten better, but after that point, the progress slows down rapidly, and very little progress is made by the 50th epoch.

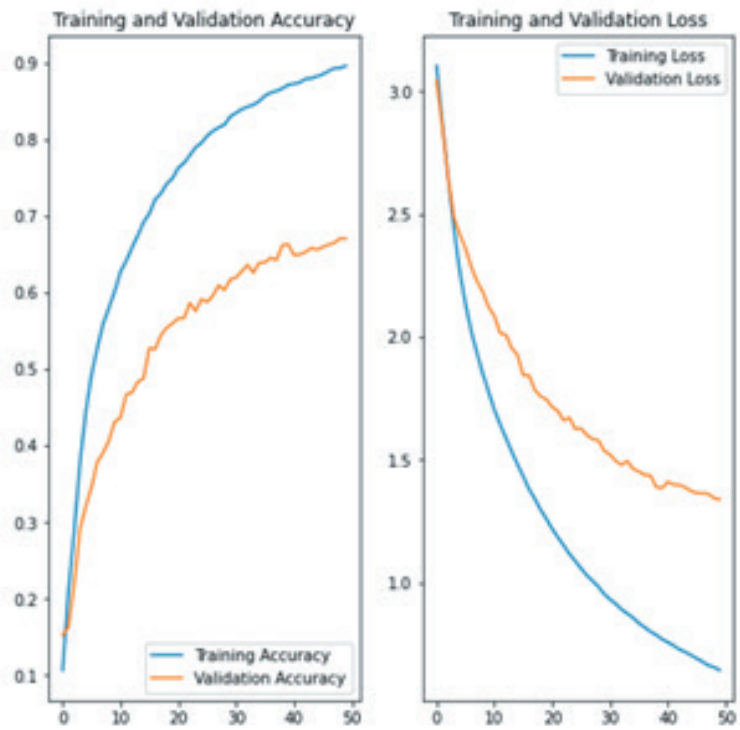

Figure 3 – Training and validation results of the custom-made CNN.

Let us see now how the transfer learning models have performed. We shall first look at the Mobile Net model. As can be seen in Figure 4, the training accuracy achieved is 83.07% and has progressed rapidly with every epoch.

Test accuracy however is very low with only 51.64% accuracy. We can see a sharp decline in learning progress by the 5th epoch and by the time all the epochs have been done, no significant progress has been made.

Figure 4 – Training and validation results of the MobileNetV2 model.

The following model results we shall discuss are of the ResNet model. As can be seen in Figure 5, the training accuracy quickly peaks at 99.92%, training loss is also very low. Validation accuracy jumps by 10% in the first 2 epochs, but it takes 8 epochs more for it to go 7 percentage points higher, capping at 86.87% accuracy. The validation loss has not gotten any better after the second epoch.

The last transfer learning model, the VGG19 has been trained in 3 epochs. As we can see in Figure 4, the model has large jumps in both training and validation accuracy for every epoch, but the validation accuracy curve quickly loses its momentum. The model achieved a training accuracy of 97.66% and a validation accuracy of 87.56%.

As we can see from the results above, the VGG19 network has the best validation accuracy of all the models used in the study. We also find it interesting that our network has outperformed the Mobile Net V2 model, which has also the worst validation accuracy at only 51.64%. An important thing that can be seen from the figures above is that the problem of overfitting is very prominent for all the models, despite the efforts of data augmentation, the dataset is simply too small to completely overcome overfitting.

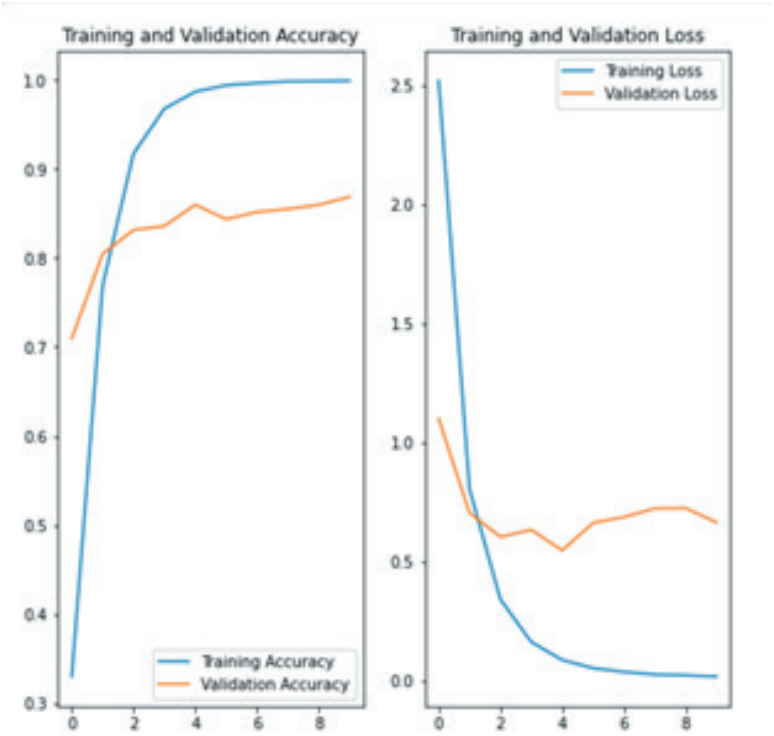

Figure 5 – Training and validation results of the ResNet50 model.

Figure 6 – Training and validation results of the VGG19 model.

5. CONCLUSION

We have seen in this study great possibilities of one day translating the entire American Sign Language. The models we have used and created have proven to be very good at successfully recognizing ASL signs with the highest accuracy we have achieved being 87.56%. A continuation of this research is required to achieve even better results. The most important piece missing in this study was a good dataset. Results can be drastically improved with a better and bigger dataset, as that would solve the biggest challenge we have faced, which is the problem of overfitting. A better dataset that would also include videos, so neural network models can be trained on gestures would be just as important. Nevertheless, convolutional neural networks have proven once again how powerful they are and the limitless number of possibilities that can be achieved with them.

6. REFERENCES

[1] N. Pugeault and R. Bowden, "Spelling it out: Real-time ASL fingerspelling recognition," 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops), Barcelona, Spain, 2011, pp. 1114-1119, doi: 10.1109/ICCVW.2011.6130290.

[2] F. Zhuang et al., "A Comprehensive Survey on Transfer Learning," in Proceedings of the IEEE, vol. 109, no. 1, pp. 43-76, Jan. 2021, doi: 10.1109/JPROC.2020.3004555.

[3] Datagen, ResNet-50: The Basics and a Quick Tutorial [Online]. Available: https://datagen.tech/guides/computer-vision/resnet-50/ [Accessed 26.03.2023.]

[4] MathWorks, MobileNetV2 [Online]. Available: https://www.mathworks.com/help/deeplearning/ref/mobilenetv2.html [Accessed 26.03.2023.]

[5] MathWorks, VGG19 [Online]. Available: https://www.mathworks.com/help/deeplearning/ref/vgg19.html [Accessed 26.03.2023.]

[6] ASL Finger Spelling Dataset, University of Exeter, Department of Computer Science, 2011. [Online]. Available: https://empslocal.ex.ac.uk/people/staff/np331/index.php?section=FingerSpellingDataset

[7] K. O'Shea and R. Nash, "An introduction to convolutional neural networks", arXiv e-prints, 2015, doi:10.48550/arXiv.1511.08458.