# American Sign Language Image Classification

1st Youval Kashuv
*Department of Computer and Information Science and Engineering*
*University of Florida*
Gainesville, USA
youval.kashuv@ufl.edu

2nd Jorge Moros
*Department of Industrial and Systems Engineering*
*University of Florida*
Gainesville, USA
jmoros@ufl.edu

3rd Rashi Ghosh
*Department of Computer and Information Science and Engineering*
*University of Florida*
Gainesville, USA
rashighosh@ufl.edu

***Abstract*—American Sign Language (ASL) is the predominant form of communication used by members of hearing impaired communities across the United States, Canada, and parts of South America, Asia, and West Africa. It is estimated that over one million individuals heavily rely on ASL. In this study, we explore the use of machine learning techniques to accurately translate images of ASL characters to English characters. We find that Convolutional Neural Networks (CNNs) are the most efficient in quickly and accurately classifying ASL characters that have been subject to rotational data augmentation techniques.**

***Index Terms*—ASL, Convolutional Neural Networks, supervised learning, image prediction**

## I. Introduction

Although ASL is the predominant form of communication used by communities across the United States, of the forty-eight million individuals who suffer from hearing loss across the nation, less than one percent can sign and understand ASL. The ability to quickly and accurately translate ASL is a crucial step in forming a way for every day people to communicate with those with hearing impediments. Language is the foundation on which society is built and a bridge connecting ASL to English in real time is essential in an evolving society. One such way of building this bridge is through the use of deep learning and transfer learning.

Deep learning techniques, most commonly referred to as Artificial Neural Networks (ANNs), have seen tremendous success in both supervised and unsupervised tasks. Their ability to learn intricate, complex, and nuanced features within data make them a powerful tool for regression and classification [5]. Convolutional Neural Networks, a type of ANN, have proven to be most effective on images [6]. We show through experimentation and testing that CNNs are both effective at classifying images of ASL symbols subject to rotational augmentation and achieve higher accuracy than their counterpart, support vector machines (SVMs).

We also employ transfer learning to adapt a pre-trained model (ResNet50) to our specific dataset, leveraging its learned features and patterns to achieve higher accuracy and efficiency in our task. Transfer learning can both save time and achieve higher accuracy than a model designed from scratch [7].

## II. Implementation

Our model is built on top of ResNet50, a convolutional neural network (CNN) that is fifty layers deep. In this section, we will discuss the structure of ResNet50. Then, we will explain how we utilized transfer learning to incorporate ResNet50 into our final model implementation.

### A. ResNet50

ResNet50 is a 50 layer deep CNN which has obtained worldwide recognition for both its depth and efficiency [2]. It features convolutional layers, pooling layers, and fully connected layers. ResNet50 predominantly uses 3x3 filters with batch normalization and ReLU activation following each convolution.

The key innovation is ResNet50's success is its use of *residual blocks*. As an ANN gets deeper, it becomes evermore difficult for the gradient of the loss function to backpropogate through every layer. However, the use of residual layers have been shown to mitigate this vanishing gradient problem. In essence, a residual block is a stack of layers in which the output of one layer is also taken into account in another layer, deeper within the block. Mathematically speaking, define $H(x)$ from some input $x$ to be the desired underlying mapping for a few stacked layers. Then the layers within this block learn the residual function $F(x) = H(x) - x$. Instead of learning $H(x)$ directly, these layers learn $F(x)$ and later add x back to it. As mentioned earlier, the use of residual blocks helps in addressing the vanishing gradient problem, making a deep network more reliable and easier to train [1].

### B. Transfer Learning Model

Now that we have covered ResNet50, we can take a closer look into our model. Our model uses ResNet50 as a base model for transfer learning followed by a 2D global average pooling, and two dense layers.

*1) Defining the Base Model:* To begin, the ResNet50 base model had to be initialized. The *pre-trained weights* were set to `imagenet` in order to use information captured from the ImageNet dataset in our model.

The final fully connected layers of the ResNet50 model were set to `false`. This is common practice for transfer learning, so that later we can stack our own layers on top to train our specific model.

The input shape of the base model was set to match the dimensions of our dataset: 300 pixels x 300 pixels x 3 channels (for RGB).

Finally, the base model was specified to be trainable. Usually during transfer learning, the pre-trained weights are "frozen" and are not updated when training on a new dataset. However, allowing the base model to be trainable (ie, the pre-trained weights will be updated) allows us to fine tune the pre-trained weights on our ASL dataset.

*2) Defining our Model:* The next step was to define our model on top of the base model. We used Keras's Sequential API to define a sequential model.

The first set of layers consisted of the aforementioned base ResNet model. This set of layers serves as a feature extractor. Then, we added a *global average pooling layer*. This pooling strategy is related to *average pooling*. In average pooling, a fixed size window is fit over regions of the feature map, and the average of each region is computed. The goal is to reduce the dimensions of the input (thereby reducing the number of parameters). With *global average pooling*, the average of the *entire* feature map is computed, resulting in a single value. This greatly reduces the number of parameters and can avoid overfitting. Global average pooling layers are often used as the final layer of a CNN. In our implementation, this layer averages all the values of feature map output from the ResNet base model, and reduces it down to a single value.

After feature extraction, we proceeded with defining a multi-layer perceptron. A dense layer with 1024 units and the ReLU activation function was added. This dense layer also had an L2 regularizer to keep the weight values small in order to mitigate any overfitting. After this layer, batch normalization was performed in order to stabilize and speed up training. A dropout layer with 50% dropout was added – another technique to mitigate overfitting by randomly "shutting off" 50% of the units. Finally, the output layer consisted of 9 units, each corresponding to one of the 9 ASL letters. The L2 regularizer term was added to this layer. The softmax activation function was selected for the output layer to predict class probabilities.

We also created a separate model that can deal with images the model is unsure of (such as a blurry image or one not even of an ASL letter). This model had the same architecture except the final layer consisted of 10 units, one for each of the 9 ASL letters and one for unknown. Additionally, we decided on defining a threshold of 0.7 for the softmax activation. That is, if the model is not at least 70% sure of its prediction for any class, the image get classified as unknown. This model was to be tested on a hard test set that consisted of unknown images.

## III. Experiments.

A few experiments were conducted in order to determine (1) the model architecture and (2) the best hyperparameters for our final model implementation. This section details the experimentation process.

*1) Preparing the Data:* This paper focused on classifying only the first nine letters of the ASL alphabet, A-I. The data was collected by students in the course, where students took pictures of themselves signing ASL letters A-I. The final dataset consisted of images that were rotated and had noisy backgrounds.

The final dataset used for our model contained 8443 samples, each image with dimensions 300x300x3. We had 9 labels, with each label representing an ASL letter A-I. Figure 1 illustrates some samples from our dataset.

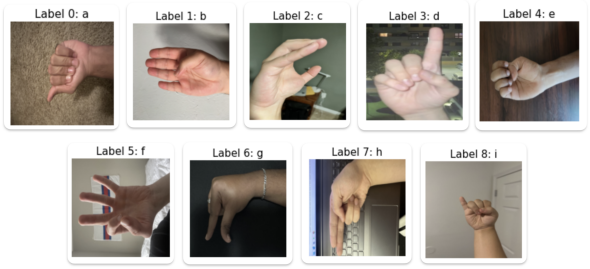


Fig. 1. Samples from the dataset

For parameter experimentation, the data was first randomly split into training and test, with 20% of the data in test, and 80% in training. The training set was further randomly split into train and validation, with 70% of the data in train, and 30% of the data in validation. All test labels were one-hot encoded.

*2) Model Architecture Selection:* Initially, a support vector machine (SVM) was experimented with to implement classification of sign language images. The first implementation consisted of a basic soft margin SVM architecture and yielded very low accuracy scores on the test set ($\leq$40%). The primary challenge was the prevalence of *rotated images* in our dataset. To address this, we attempted *feature extraction* using the Histogram of Oriented Gradients, or HOG, feature descriptor [3]. HOG is a powerful feature extraction technique in computer vision and image processing that analyzes the orientations of an object's edges to describe its shape and appearance [4]. Scikit learn provides a package for HOG feature extraction, in which parameters such as orientations, cells per block, and pixels per cell were experimented with. After HOG feature extraction, the new features were fed into the SVM model. This approach achieved 86% accuracy on the test set.

Artificial neural networks – especially convolutional neural networks – combined with transfer learning appeared to be more promising due to available pre-trained CNN's on much

larger datasets. This approach would allow us to adapt existing architectures trained on a much larger datasets, and fine-tune them to our specific dataset of ASL images.

*3) Hyperparameter Tuning:* Our model consisted of several hyperparameters. Various values were selected and evaluated based on the following criteria: (1) accuracy on the validation set, and (2) availability of computational resources.

The following hyperparameters were experimented with using the specified values to fine-tune the model's performance while minimizing computational cost:

- Optimizer (`Adam`, `Nadam`)
- L2 Regularization Term (`1e-3`, `1e-5`, `1e-7`)
- Initial Learning Rate (`1e-3`, `1e-4`, `1e-5`)
- Learning Rate Reduction Factor (`0.01`, `0.1`)
- Early Stopping Patience (`5`, `10`, `50`)
- Batch Size (`None`, `16`, `32`, `64`)
- Number of Epochs (`20`, `50`)

For the *optimizer*, two options were considered: the Adam optimizer and the Nadam optimizer. The Adam optimizer allows for an adaptive learning rate and momentum, helping to achieve smoother training. Nadam is nearly identical, with the exception that it uses Nesterov's formula for the momentum term, which adds the correction before computing the gradient for the gradient descent update rule. Between the two, `Nadam` yielded better accuracy on the validation set.

For the *L2 regularization term* (which was used in the dense layers), a value of `1e-5` yielded best accuracy on the validation set. For the *initial learning rate*, a value of `1e-4` was determined to be optimal. The *learning rate reduction factor* hyperparameter allowed us to have more control over how much the learning rate would decrease by. We selected a value of `0.1` based on validation accuracy and computational resources.

For *early stopping patience*, `10` epochs was found to be optimal. For a lower value, the loss during training was still decreasing, and for higher values, the loss was not decreasing. For a *batch size* of `16` and *number of epochs* set at `20`, the training reached the maximum epochs and the loss was still decreasing. Adjusting the *number of epochs* to `50` allowed the model to converge, and resulted in optimal validation accuracy. We found `50` epochs to be sufficient for the model to converge with negligible change in loss.

The outcomes of the experimentation resulted in the following tuned hyperparameter values:

- Optimizer: `Nadam`
- L2 Regularization Term: `1e-5`
- Initial Learning Rate: `1e-4`
- Learning Rate Reduction Factor: `0.1`
- Early Stopping Patience: `10`
- Batch Size: `16`
- Number of Epochs: `50`

## IV. Results

In this section, we will discuss the results from implementing our model with the tuned hyperparameters detailed in the previous section.

### A. Training.

*Compiling the model*. To train the model, the `Nadam` optimizer was used with a very small initial learning rate of `1e-4`. The Adam optimizer allows for an adaptive learning rate and momentum, helping to achieve smoother training. For the loss function, `categorical cross entropy` was selected. This loss function is appropriate for multi-class classification tasks in which the labels are one-hot encoded. The metric reported from training was `accuracy`.

*Callbacks.* During training, three callbacks were defined. The first was `early_stopping`, which helps mitigate overfitting. The `patience` and `monitor` parameters were set such that if validation did not improve for 5 epochs, training would be stopped. The callback was also configured such that the model's weights would be restored to the best values observed before stopping.

The second callback, `reduce_lr`, helped control the adaptive learning rate. This callback was configured such that when validation loss stopped improving for 5 consecutive epochs, the learning rate was reduced by a factor of 0.1. This helped promote smoother convergence when approaching a minimum in the loss function.

The final callback, `checkpoint`, was responsible for saving the best model in training (to be later used during test). This callback was configured such that only the best model's weights (based on validation loss) was saved in a file named `best_model_resnet.h5`.

*Fitting the Model.* Finally, the model was fit using the train and validation data (details specified in the *Preparing the Data* subsection). We specified 50 epochs in batches of 16 samples, and included all three callbacks.

### B. Results.

The resulting model was evaluated on the test set (20% of the data, unseen). The test accuracy score was 98.11%. Figure 2 graphs the learning curves from training. It is worth noting that the validation loss curve is not too smooth and rather jagged, which could be due to the smaller batch size. The smaller the batch size, the more "influence" the batch has on the learning curve, pulling it in more directions between epochs.

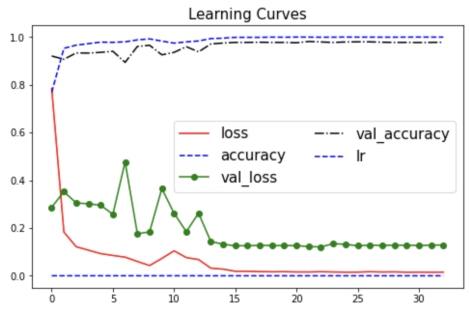


Fig. 2. Learning curve during training. Percentages on Y-axis, number of epochs on X-axis

Figure 3 breaks down the understanding of our models' performance by exploring precision, recall, and F1 score. *Precision* focuses on the percentage of true positives by examining: of all the positive class predictions, how many were actually correct? For all classes, our model achieved high accuracy scores of over 95%, meaning when our model predicts positive for a class, it's likely to be correct. Our model also achieved 95% or above for *recall*, which measures the model's ability to detect all positive instances of a class. Finally, the *F1 score* provides a balanced measure between precision and recall. The high F1 score (also over 95% for all classes) indicates our model strikes a good balance, which is consistent with the individual precision and recall scores.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.98 | 0.98 | 0.98 | 175 |
| 1 | 0.99 | 0.99 | 0.99 | 177 |
| 2 | 0.97 | 0.97 | 0.97 | 155 |
| 3 | 0.99 | 0.98 | 0.99 | 193 |
| 4 | 0.98 | 1.00 | 0.99 | 193 |
| 5 | 1.00 | 0.99 | 0.99 | 203 |
| 6 | 0.97 | 0.95 | 0.96 | 189 |
| 7 | 0.96 | 0.98 | 0.97 | 198 |
| 8 | 0.98 | 0.98 | 0.98 | 206 |
| accuracy | | | 0.98 | 1689 |
| macro avg | 0.98 | 0.98 | 0.98 | 1689 |
| weighted avg | 0.98 | 0.98 | 0.98 | 1689 |

Fig. 3. Model performance during test

Finally, Figure 4 provides a visual of correct and incorrect classifications. Overall, the darker diagonal on the confusion matrix indicates the model is pretty accurate at predicting the correct letter for the sign language images. This confusion matrix also reveals that class 7, or the letter "I", is the most mis-predicted letter (8 total misclassifications). The model mostly confuses this letter with class 6, or the letter "G".

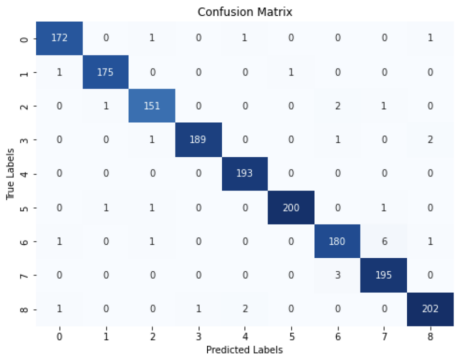


Fig. 4. Confusion matrix of model predictions during test

## V. Conclusion

Image classification on American Sign Language (ASL) images can be achieved by implementing transfer learning. This task proved to be incredibly challenging at first due to our smaller yet noisy dataset, with a lot of images with background noise of different rotations. However, convolutional neural networks and transfer learning provided an elegant solution for this problem. Using ResNet50, a convolutional neural network pre-trained on a very large image dataset, we were able to implement a transfer learning model to accomplish our task. We employed overfitting mitigation techniques such as L2 regularizer and dropout layers, and experimented with various hyperparameter values in order to appropriately tune them. Ultimately, our model achieved over 98% accuracy on an unseen test set, a balanced F1 score for all classes, and relatively few misclassifications overall.

Our model still has room for improvement. Future work includes classifying the remaining 17 characters. Additionally, data augmentation techniques could be implemented to solidify our model. Finally, we only implemented a single hidden layer in our multilayer perceptron; it is worth investigating multiple layers and experimenting with different numbers of units.

That being said, our work provides a solid basis for implementing a machine learning model that can classify ASL images with high accuracy.

## References


[1] H. Yadav, "Residual Blocks in Deep Learning," Medium, Jul. 11, 2022. https://towardsdatascience.com/residual-blocks-in-deep-learning-11d95ca12b00 (accessed Dec. 03, 2023).

[2] "ResNet-50: The Basics and a Quick Tutorial," Datagen. https://datagen.tech/guides/computer-vision/resnet-50/

[3] M. Tyagi, "HOG(Histogram of Oriented Gradients)," Medium, Jul. 24, 2021. https://towardsdatascience.com/hog-histogram-of-oriented-gradients-67ecd887675f

[4] A. Singh, "Feature Engineering for Images: A Valuable Introduction to the HOG Feature Descriptor," Analytics Vidhya, Sep. 04, 2019. https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/

[5] Mohammad Mustafa Taye, "Understanding of Machine Learning with Deep Learning: Architectures, Workflow, Applications and Future Directions," Computers, vol. 12, no. 5, pp. 91–91, Apr. 2023, doi: https://www.mdpi.com/2073-431X/12/5/91.

[6] R. Yamashita, M. Nishio, Richard Kinh Gian, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," Insights into Imaging, vol. 9, no. 4, pp. 611–629, Jun. 2018, doi: https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9.

[7] K. R. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," Journal of Big Data, vol. 3, no. 1, May 2016, doi: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-016-0043-6.