Sign Language Gesture Recognition using Bayesian Optimization and Transfer Learning

Sakshi Sangtani
University of Potsdam
sangtani@uni-potsdam.de

Bashar Arous
University of Potsdam
arous@uni-potsdam.de

Md. Abdul Aziz
University of Potsdam
aziz@uni-potsdam.de

Afnan Ansar Shaikh
University of Potsdam
afnanansarshaikh@uni-potsdam.de

Pradum Chauhan
University of Potsdam
chauhan2@uni-potsdam.de

Abstract

Speech impairment is a physical disability that impairs a person's capacity to communicate verbally and audibly. People who are impacted by this employ sign language and various alternative forms of communication. In this case, the use of hand gestures is the most common way of communication for the deaf, particularly in American Sign Language (ASL), which is used to represent the alphabet, the numbers, and also a lot of times, complete words. To help the deaf communicate with everyone, the community needs a system to overcome the barrier between the people who are not hearing-impaired and the people who are. Several studies on the challenges with sign language translation have been carried out. Our research suggests a model for translating the ASL alphabet from static images, using Deep Learning methods. The suggested approach starts with some pre-processed images taken from a webcam. To classify the gesture as a letter, we used a Convolutional Neural Network (CNN) architecture to create a base model. Along with that, we apply transfer learning on the same dataset using models from Keras that are already trained on the ImageNet dataset.

1 Introduction

For the deaf, sign language is the only form of communication that uses various signs made by moving the hands in conjunction with facial expressions and body postures. Hand gesture recognition is one of the areas where image processing techniques and computer vision can be used to help the deaf community.

Several types of research focus on sign language translation issues to help the deaf easily communicate with people who do not have hearing impairments. Sign language has broad categories that include, both, local and universal languages. There are numerous ongoing studies on automatic local sign language translation, including Indian, Malaysian, and Thai sign languages. Among all, American Sign Language (ASL) (See Fig. 1) is one of the most important and universally adopted languages for the deaf. Consequently, many researchers are emphasizing for the development of automatic ASL translation systems [1].

Therefore, motivated by the above reasons, we propose a system capable of automatically recognizing the alphabet, i.e the 26 letters. In this work, we worked on developing our own model using Convolutional Neural Network and hyperparameter optimization. We then compared it with a bunch of infamous pre-trained models by using the concepts of transfer learning, on the same dataset that was used for the base model.

In this paper, we start with talking about the related work in the Section 2. The Section 3 talks briefly about the dataset we use, and the augmentations we have implemented on it. Section 4 goes deeper into the explanation of the Model Architecture and Training, as we talk about our own CNN model, the Hyperparameter Optimization Process, and also talks about the transfer learning process. Section 5 goes in the details about the model performance, by itself, and in comparison to the pretrained models. Section 6 discusses the experiments with the dataset, as well as possible improvements. Section 7 talks about our Future Work.

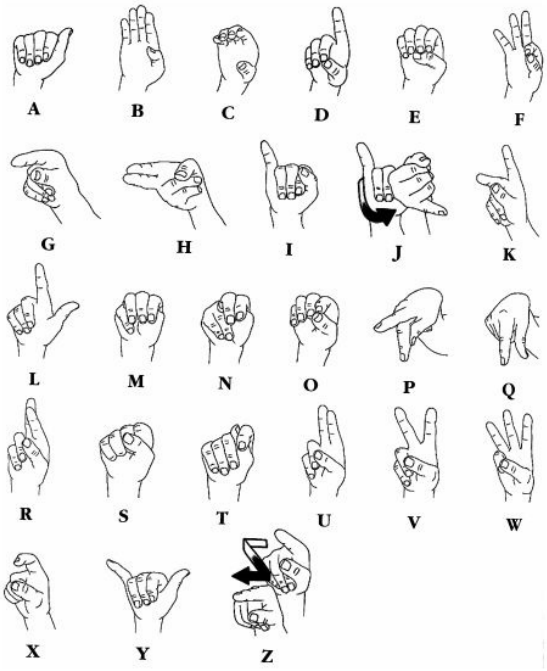


Figure 1: American Sign Language

2 Related Work

In recent years, several deep learning models have been used to solve the problem of translating sign language into text. For instance, [2] employs Inception and Convolutional Neural Networks (CNN) for identifying spatial characteristics, and Recurrent Neural Networks (RNN) for training on temporal information.

A modern method for recognizing ASL letters has been proposed in [3] which uses a low-cost depth camera called Microsoft's Kinect. 24 ASL static alphabet signs could be recognized using this approach, with roughly 90% accuracy.

A method has been developed in [4] to automatically generate speech for the findings of hand sign-based digit detection in Bangla. With a 92% accuracy rate, the CNN-based deep learning model was used to build the system's core.

A deep neural network-based classifier that categorizes, both, the images of letters and numbers in ASL, has been published in [5]. It achieves 82% accuracy on the alphabet gestures, and 97% accuracy on the validation set for numbers.

With a self-created dataset of 1200 samples of 10 static signs or characters, Chen [6] proposed a model. Edge segmentation was used as the first step in pre-processing to identify the edges of the hands to understand the motions, and skin colour segmentation was accomplished by converting RGB pictures to YUQ or YIQ colour spaces [6]. The convex hull approach was then used to identify the fingers from the hand that had already been recognized. The classification approach was finally implemented using neural networks. This model's ultimate accuracy score was 98.2%.

Author	**Methodology**	**Dataset**	**Accuracy**
Mittal et al. (2019)[7]	2D-CNN and Modified LSTM, with Leap motion sensor	ASL	89.50%
Aparna and Geetha[8]	CNN and 2layer LSTM	Custom Dataset (6 signs)	94%
Jiang et al. (2021)[9]	3DCNN with SL-GCN using RGB-D modalities	AUTSL	98%
Liao et al. (2019)[10]	3D-ConvNet with BLSTM Inflated 3D ConvNet	DEVISIGN D	89.8%
Adaloglou et al.(2021)[11]	with BLSTM	RGB + D	89.74%

Table 1: Comparative analysis of various deep learning models/methods.

According to a method presented by Camgoz et al. [12], the translation of sign language into spoken language is not a one-to-one mapping but rather a sequence-to-sequence mapping. The tokenization and embedding processes of the common neural machine translation were replicated in the authors' newly developed vision technique. The CNN architecture [12] is integrated with an attention-based encoder and decoder that predicts the probabilistic likelihood of generating a spoken language from a given signing video in the neural machine translation stage, which converts sign videos to spoken language. Starting with word embedding, the authors converted a sparse vector into a denser form where words with related meanings were closer together. A fixed-size vector of the features in the sign videos was produced during encoding. The previously hidden state and the word embedding from the decoding stage served as the inputs. This made it easier to guess the subsequent word. The authors also included an attention mechanism in the decoding phase to address the issues of long-term dependencies and disappearing gradients.

Our goal is to achieve better accuracy than previous works, and to try different architectures to compare the performance of the models.

3 Dataset

In this project, the data used was taken from a publicly available dataset [13] from Kaggle. The dataset was created by a contributor on Kaggle, with the intention and purpose of using Deep Learning for Sign Language Gesture Recognition. In this dataset, the images were taken with the help of a laptop web camera and the collection is separated into 29 folders representing various classes.

3.1 Dataset Description

Since not everyone has a professional camera or environmental setting, this dataset feels more realistic because of the different lighting conditions and angles used for generation. As mentioned in the dataset overview, this dataset included 29 classes, three of which are 'Space', 'Delete', and 'Nothing', and the remaining 26 are the letters of the alphabet. There are a total of around 87000 images in the training dataset which are 200×200 pixels. As we see in the dataset in Figure 2, the image quality and different environment settings, are similar to that of a real-life photograph of a hand. This helps us create a more reliable model. Thus, this dataset was used to develop the base model.

We also use the terms “base model dataset” and “webcam dataset” for this dataset, interchangeably in this paper.

Figure 2: Preview of Base Model Dataset

3.2 Data Augmentation

We use Data Augmentation during the data pre-processing. The objective here is to essentially increase the fidelity of the dataset in an artificial sense. We use it only on the training data. The validation and test datasets were only rescaled without any special augmentation parameters.

For the base model dataset, the techniques used were re-scaling, and random horizontal flipping and rotation of the images. In our case, these were done to make the images look more realistic and also to account for left- and right-handed gestures. (See Fig. 3).

Figure 3: Data Augmentation on Base Model Dataset

3.3 Data Split and Classes

In this project, we just concentrate on the alphabet classes and excluded the other 3 classes, thus making it an image classification task with 26 classes from A-Z. For the base model, the input and output image dimensions are resized to 64×64. This gives us a dataset of 78026 images. For the training-validation-testing split, the images was split in the ratio of 75:15:10. When we apply transfer learning with the pretrained models, we use a 128×128 image size due to the minimum requirements of the pretrained models.

4 Model Architecture and Training

In the current section, the model architecture and the training process will be outlined. A *Convolutional Neural Network*, or *CNN*, is a kind of artificial neural network. CNNs have fundamentally altered the approach to image recognition because they can detect and interpret patterns. They are one of the most effective architectures for image classification, recognition, and detection tasks due to the high accuracy of their results.

4.1 Base Architecture Layers

Some of the CNN layers that our model includes are - MaxPooling, Batch Normalization, Dropout and then, of course, to add non-linearity, the activation functions. MaxPooling helps us in extracting low-level features from the images, thus down-sampling the input’s spatial dimensions. Batch Normalization allows every layer to learn independently by normalizing the output of the previous layers. We use the Dropout layer in this case, to prevent overfitting of the model. Using a dropout rate of 0.5 was found to give the maximum amount of regularization [14]. Since it is a multi-class classification, using the Softmax activation function in the output layer was the optimal choice.

4.2 Base Model and Training

For the base model of the project, (See Fig. 4), a sequential model is used to which a 2D Convolution Layer is added for several layers. The model depth, i.e. the number of layers in the model, is optimized based on the test evaluation results. (See Section 4.3.2) During the Hyperparameter Optimization, the number of layers are optimized along with other hyperparameters, to determine the ideal balance between model complexity and accuracy. This method touches the foundations of what is called *Neural Architecture Search* (See Section 4.3.3). The shape of the input was set to $64 \times 64 \times 3$, the dimensions of the *kernel* to 5×5, and the spatial dimensions are preserved by keeping the *padding* the same. The optimizing sequence also included optimizing the activation functions, with the options of ReLu, SeLU and ELU. *Max Pooling*, *Batch Normalization*, and a *Dropout* of the rate of 0.5 are applied for all layers, except for the final layer. We then add a fully-connected layer at the end of the network. The multidimensional input has to be *flattened* into a single dimension, to which a dense layer with *softmax* activation is added. The model is then compiled with the *Cross Entropy Loss* (See Eq. 4) and *Adam* (Adaptive Moment Estimation) as the optimizer. We also implement an Early-Stopping Callback function, that stops the model training if there seems to be no improvement in the metric that is being monitored. In our case, we evaluated the validation loss during the model training, and if the validation loss had not improved for more than 5 epochs, the model would stop training at that point.

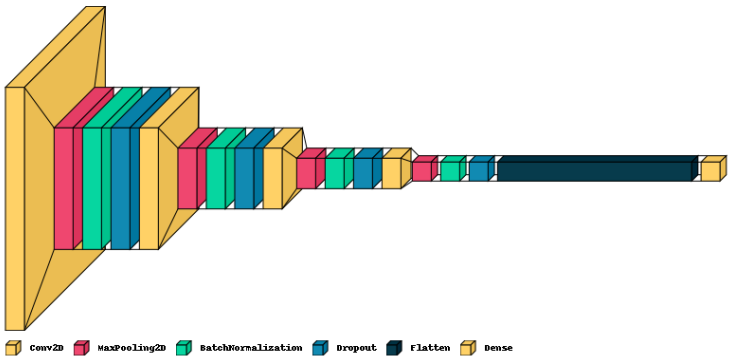


Figure 4: Base Model Architecture

Adaptive learning rates are calculated for each parameter using the Adam method. Adaptive Gradients(AdaGrad) and Root Mean Square Propagation(RMSProp) are two stochastic gradient descent techniques that are combined in ADAM. This optimization algorithm computes the gradient based on a randomly selected subset of data, rather than using the entire data set.

4.3 Hyperparameter Optimization

As defined by Morozov, Anton D., et al. [15] "Hyperparameter Optimization is the problem of choosing a set of optimal hyperparameters for a learning algorithm". This plays a very important role in prediction accuracy and is also considered the trickiest part of building models[16]. The objective function takes in hyperparameters and outputs a single real-valued score that we want to minimize (or maximize).

Before model training, the hyperparameter tuner, which is external to the model, is tuned. The ideal hyperparameter values that come from the tuning procedure are then fed to the model training stage. Our hyperparameter optimization process is illustrated in Figure 5.

To optimize hyperparameters, techniques like Grid Search and Random Search are frequently utilized (See Fig. 6). Grid Search uses straightforward brute force to find the optimum hyperparameters. In this, we take values of the hyperparameters(essentially creating a search space) at discrete, equally sized steps, which are then implemented in the model and tested as individual combinations. Hyperparameters from the search space are sampled at random by Random Search, and it outperforms grid search in, both, theory and practice. This means that finding the ideal hyperparameter setup takes less time and effort.

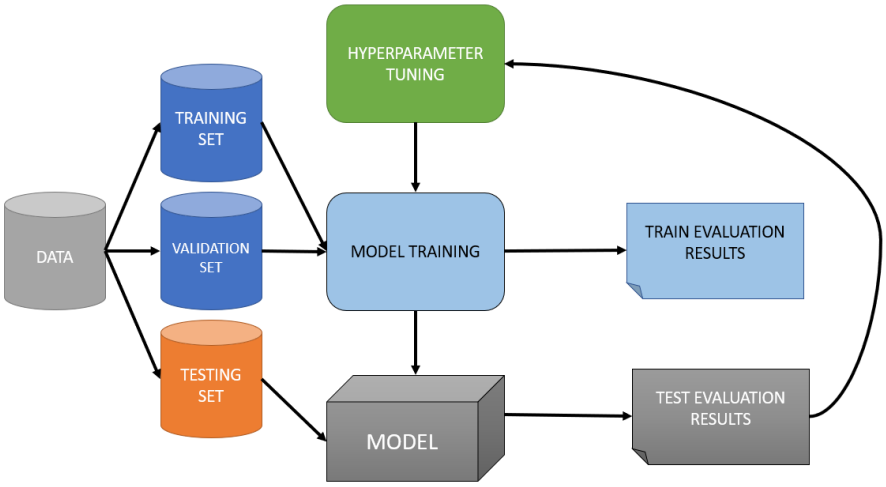


Figure 5: Model Selection.

But these two methods are inefficient at sampling the search space. Today's deep learning algorithms frequently include a lot of hyperparameters, and training a good model might take days or even weeks. If there is a high number of hyperparameters along with a really big range of values each, it is simply not viable to train different models for each hyperparameter combination by brute force. Of course, in cases of really small search spaces, these methods are pretty efficient and work quite well.

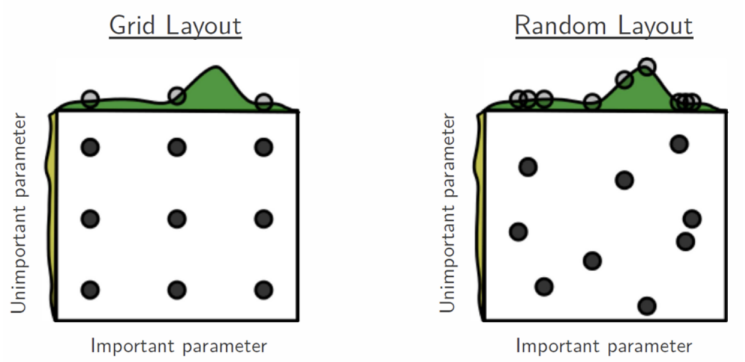


Figure 6: Grid and Random Layout.

4.3.1 Bayesian Optimization

Contrary to random or grid search, Bayesian techniques maintain a note of the previous assessment results, which they employ to generate a probabilistic model that links hyperparameters to the probability of a score on the objective function[17]:

$$P(score|hyperparameters) \tag{1}$$

This model, represented as $P(y|x)$, is known as a "surrogate" for the objective function[18]. The objective function is far more difficult to optimize than the surrogate function, and Bayesian approaches determine the next set of hyperparameters to test the objective function by choosing the best-performing hyperparameters for the surrogate function.

For this project we used, *Optuna*, a hyperparameter optimization framework that uses Bayesian Optimization with the state-of-the-art *TPE Sampler*.

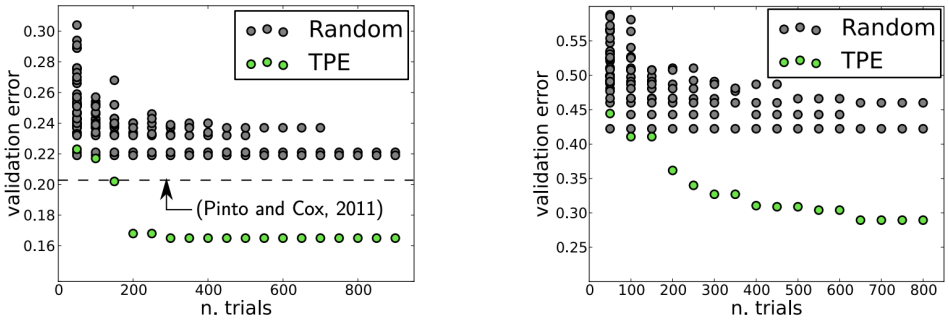


Figure 7: Comparison of validation error for hyperparameter optimization with random search.

With the random search in grey and Bayesian optimization (using the Tree Parzen Estimator or TPE) in green, these figures (See Fig. 7) compare validation errors for hyperparameter optimization of an image classification neural network. Lower is better: fewer trials imply less time spent testing, and smaller validation set errors often correspond to higher test set performance [19].

The Optuna framework[20] is used to automate the optimization of these hyperparameters. By using various samplers, such as grid search, random, bayesian, and evolutionary algorithms, it automatically chooses the ideal hyperparameter values. Optuna's primary features include automated hyperparameter search, large-space search efficiency, trial pruning for quick results, and parallelizing hyperparameter searches over multiple threads or processes. By combining sampling and pruning techniques, Optuna offers effective hyperparameter tuning.

The standard sampler in Optuna is the Tree-structured Parzen Estimator(TPE). It samples the following hyperparameter settings using the historical data of earlier assessed hyperparameter setups.

The Tree-structured Parzen Estimator builds a model by applying the Bayes rule. Instead of directly representing $p(y|x)$, it instead uses:

$$P(\text{y} \mid \text{x}) = \frac{P(\text{x} \mid \text{y}) \times P(\text{y})}{P(\text{x})} \tag{2}$$

$P(x|y)$, which is the probability of the hyperparameters given the score on the objective function, in turn, is expressed:

$$P(x|y) = \begin{cases} l(x), & \text{if } y < y^* \\ g(x), & \text{if } y \geq y^* \end{cases} \tag{3}$$

where $y < y^*$ signifies an objective function value that is less than the threshold. And $y \geq y^*$ signifies an objective function value that is greater than the threshold[21].

4.3.2 The HPO Process

In this project, the data is split into training-validation-testing, as mentioned before. For the model selection during the trials of the optimization algorithm, the test data evaluation results are used as a deciding factor for the best model, i.e. base model to be chosen by the Hyperparameter Optimization Algorithm. This is because we use the callback function called Early-Stopping, which is dependent on the loss of the validation set. The model selected at the end, is thus, not overfitting on the training or the validation set, and there is no risk of a data leak.

As we see in Table 2, the image size and number of epochs are fixed while the other parameters are being optimized.

There were a total of 25 trials. Every trial was trained for 30 epochs, or until there was no model improvement, in which case there was early stopping. The table below (Table 3) shows a preview of the HPO Study results. We can see that some of the trials give us really good results. Looking at

Hyperparameter	Optimized	Values for Trials	Final Value
Image Dimensions	False (Fixed value)	-	(64,64,3)
Epochs	False (Fixed value)	-	30
Batch Size	True	Range (64 to 128)	105
Number of CNN Layers	True	(3,4,5,6)	4
Activation Function	True	ReLu,SELU,ELU	ELU
Learning Rate	True	Range (1e-5 to 1e-2)	0.001698893

Table 2: Hyperparameters Values

Trial 6, 7, and 9, we can infer that we could have had more than one configuration of hyperparameter values that could work well for our final model. Trials 1 and 16 give extremely low accuracy, which seems like it could have been caused by the really low learning rate. The Trial 23 was our best trial, whose hyperparameter values was then used to create our base model.

Trial Number	Activation Function	Batch Size	Learning Rate	Number of CNN Layers	Duration (mm:ss)	Test Accuracy
1	selu	89	0.000033601	4	11:36	13.85%
6	selu	86	0.000749395	6	20:30	98.20%
7	selu	107	0.004528772	3	35:43	99.58%
9	relu	108	0.000156515	5	41:03	99.11%
16	elu	64	0.000011751	5	08:13	4.08%
23	elu	105	0.001698893	4	33:53	99.93%

Table 3: Some trials from the HPO Study

4.3.3 Neural Architecture Search

The goal of Neural Architecture Search is to find the best neural network architecture for a specific problem [22]. An abstract illustration of the methods under NAS can be seen in Figure 8. It is one of the most rapidly growing areas of machine learning. We implement the core concept of Neural Architecture Search(NAS) in our Hyperparameter Optimization method, as we try to optimize the depth of our model architecture. Our final model had 4 layers.

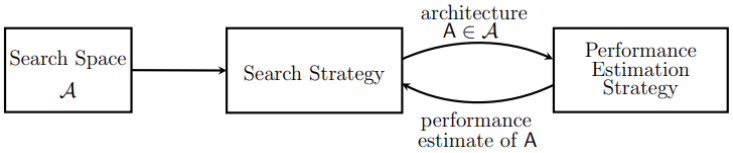


Figure 8: Abstract illustration of Neural Architecture Search

4.4 Transfer Learning using Pretrained Models

A deep CNN model may take days or weeks to train on very large datasets[23]. Thus, we use the concept of Transfer Learning. We take the weights of models that have been already trained on a significantly huge dataset, which is in this case, a set of models that have been trained on the ImageNet dataset[24].

One or more layers from the pre-trained model are used in a new model being trained on a problem of interest. This process can also result in lower generalization errors. The weights in the re-used layers are used as the starting point for the training process and adapted to the new problem. We focus on four of the more popular models, that include variations of *VGG*,[25] *Inception modules*[26], *Xception modules*[26], and *residual modules (ResNet)*[27]. We compare the results in the Section 5.3

5 Model Evaluation and Prediction Results

5.1 Metrics

We used the categorical cross-entropy loss and accuracy metrics for the model evaluation. For the prediction results, we always generate a Confusion Matrix and Classification report to understand the Type I and Type II errors.

Cross Entropy Loss - The categorical cross-entropy loss function estimates the loss of an example by calculating the following sum:

$$Loss = -\sum_{i=1}^{\text{output size}} y_i.log\hat{y}_i \tag{4}$$

where output size is the total number of scalar values in the model output, $\hat{y_i}$ is the i-th scalar value in the model output, and y_i is the associated target value. This loss is an effective measuring device to calculate how easily two discrete probability distributions may be distinguished from one another.

Confusion Matrix - A confusion matrix (See Table 4) is one of the metrics to analyze the performance of a classification algorithm. It consists of four basic characteristics which are True Positives (predicted value is positive and its true value is positive), True Negatives (predicted value is negative and its true value is negative), False Positives or Type I error (predicted value is positive but its true value is negative) and False Negatives or Type II error (predicted value is negative but its true value is positive).

		Predicted Values		
		Positive	Negative	Total
Actual Values	Positive	TP	*FN(Type II error)*	$TP + FN$
	Negative	*FP(Type I error)*	TN	$FP + TN$
	Total	$TP + FP$	$FN + TN$	*sum total*

Table 4: Confusion Matrix

Accuracy - This metric describes how the model performs across all classes. It is helpful when all classes are equally important. It is evaluated as the ratio between the number of correct predictions to the total number of predictions,

$$Accuracy = \frac{\textit{True Positive(TP) + True Negative(TN)}}{\textit{True Positive(TP) + True Negative(TN) + False Positive(FP) + False Negative(FN)}} \tag{5}$$

Precision - It is used get the number of positively classified predictions that actually belong to the class. Precision is obtained by analyzing the confusion matrix, and is given by,

$$Precision = \frac{True\ Positive(TP)}{True\ Positive(TP) + False\ Positive(FP)} \tag{6}$$

Recall - It is used to obtain the number of positively classified predictions out of the total number of predictions. We infer the Recall value from the confusion matrix, and is given by,

$$Recall = \frac{True\ Positive(TP)}{True\ Positive(TP) + False\ Negatives(FN)} \tag{7}$$

5.2 Prediction Results for the Base Model

The Hyperparameter Optimization algorithm chose the parameter values based on the best trial 4.3.2 which is our base model. The parameters chosen for this model are batch size of 105, a learning rate of 0.001698893, the number of CNN layers were chosen to be 4 and ELU was the activation function (See Table 3)

We obtain a great 99.93% of accuracy after evaluating the best model on our test dataset in time duration 33:53. The training loss and the accuracy curve of our base model can be seen in Figure 9.

Thus, as is evident, that our model performs extremely well on the dataset, (See Figure 2). This is because, even with a large dataset of 78026 images, some of them are extremely similar-looking with almost no differences in between them. There were some misclassifications due to bad lighting and similar gestures. Due to the generally high similarity between images, there is very low confusion (See Figure 10) and so the misclassifications are not a lot, as even in the test data, the images would be very similar to the training set with very slight differences.

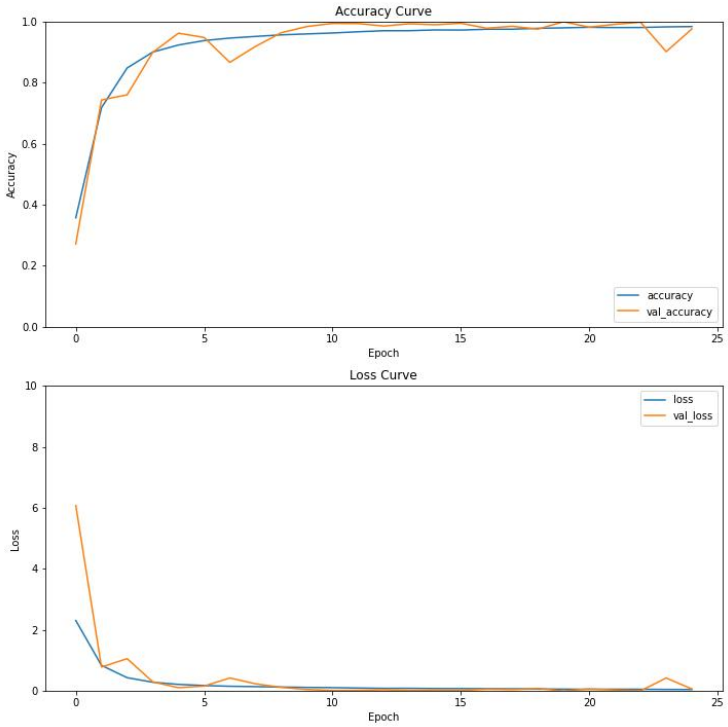


Figure 9: Training Loss and Accuracy Curve of the Base Model

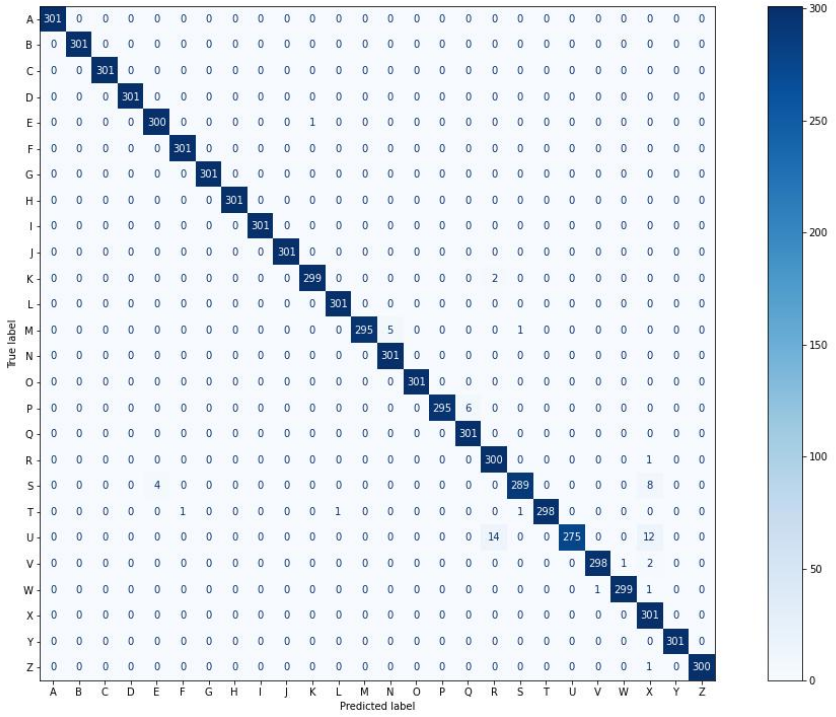


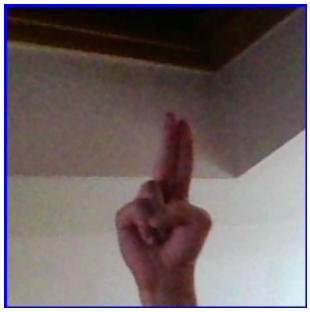
Figure 10: Confusion Matrix of the Base Model

As mentioned above, some images are misclassified due to reasons such as bad lighting, distorted angles, or even the gestures that just look similar. An example can be seen in Figures 11 and 12. The letters R and U are misclassified due to their signs being very similar (Fig. 11). Bad lighting could also play a factor sometimes. (Fig. 12). These figures justify the inaccuracies in the Confusion Matrix.

(a) R (b) U

Figure 11: Similar-looking Gestures for Letters R and U

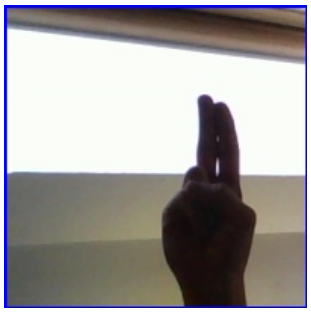
(a) R



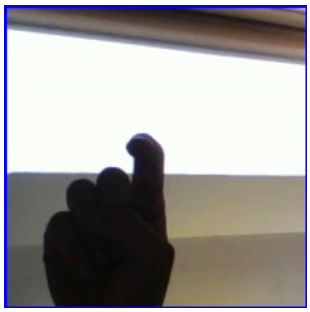
(b) U

Figure 12: Similarity Due to Low Light

Another example is of letter U and X (Figure 13), here X is being misclassified as U due to bad lighting in which the image is taken.



(a) U



(b) X

Figure 13: Signs of U and X in Low Light

5.3 Transfer Learning Results

We now want to compare the pre-trained models to see how they perform, in comparison to our base model. We use transfer learning from the models mentioned in Section 4.4 and their variations. (See Table 5) that are trained on the ImageNet[24] dataset. These models are then trained with just 10% of the total data as training set and 5% as the validation set. We train them for just 5 epochs first. The reason for only using a fraction of the total data for training and validating is because it's cost effective, and we can get a good overview more quickly and efficiently. Based on the table mentioned, the best model architecture is taken, which is the ResNet50V2, in this case. Now, a new model is created using the ResNet50V2 architecture; data augmentation is applied on the train, validation and test sets, same like it has been done for our base model initially. The model is then trained on the augmented training and validation sets with the same split as our base model, which was 75-15-10 on Train-Validation-Test. The Loss and Accuracy Curves as plotted after 15 Epochs. (Figure 14) After evaluating the model on the test data, we got a 98% accuracy.

Model	val_accuracy	Training time(sec)
ResNet50V2	85.95%	105.35
ResNet101V2	83.72%	128.45
ResNet152V2	81.70%	146.02
VGG16 [25]	79.98%	113.19
Xception [28]	76.72%	123.33
InceptionResNetV2 [29]	70.64%	165.10
InceptionV3 [30]	68.61%	110.20
VGG19	59.68%	118.80
ResNet101	25.86%	131.35
ResNet152	25.17%	148.34
ResNet50	23.76%	109.30

Table 5: Pretrained Models Accuracy after 5 epochs on Base Model Dataset

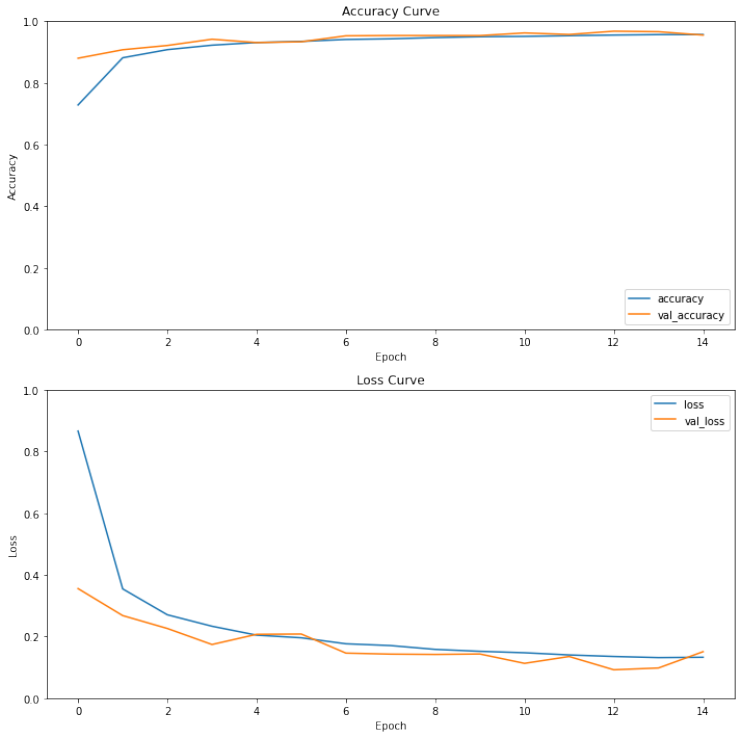


Figure 14: Loss and Accuracy Curves of ResNet50V2 after 15 Epochs

6 Discussion

Our model currently works incredibly well on our datasets, for the reasons explained above.

6.1 Experiments with the Datasets

We had a lot of trial and error in trying to find what works best. Some of the experiments we used did not go so well, but some did have really good results. The experiments were mostly done on the RTX 3060ti mobile GPU. Due to the VRAM being of only 6GB, we had restrictions on the batch size and the depth of the model architecture. Thus, when we defined the options for tuning our hyperparameters, we had to limit our search-space.

A huge part of our experiments was the Hyperparameter Optimization. Table 2 shows the range or set of values that were used for optimization. The algorithm ran for a total of 25 trials, which helped us get the best model with almost 100% (99.93%) accuracy, which we then used as the base model.

6.2 Improving the Model

The model can be more generalized by training on more datasets that look different from the current ones used. One could make the model more robust, by using Ensembling techniques. Instead of CNN's, one could experiment with, and integrate Vision Transformers as the core of the model. To make the model architecture a lot more efficient, and more advanced methods of Neural Architecture Search could be implemented.

7 Future Work

We intend to improve the model using one or more datasets that have more diversity while retaining similarities. At the moment, the proposed system can only recognize static hand postures that only account for classifying finger-spelling. In the future, we would like to classify real-time gestures using a webcam to make the system interactive and more accurate.

8 Conclusion

In our study, we develop an autonomous method for recognizing Sign Language from static photos using a publicly available American Sign Language Image dataset. We first use Hyperparameter Optimization techniques to get the best hyperparameters, along with some rudimentary Neural Architecture Search. Once we have the optimum values, we created a model using Convolutional Neural Networks, and achieved 99.93% accuracy. Then, by implementing the methods of transfer learning and fine tuning on the same dataset, we do a comparative analysis between the results from our model and the pretrained models.

References

[1] Tangsuksant, Watcharin, Suchin Adhan, and Chuchart Pintavirooj, "American sign language recognition by using 3d geometric invariant feature and ann classification.," in *In The 7th 2014 Biomedical Engineering International Conference*, pp. 1–5, IEEE, 2014.

[2] K. Bantupalli and Y. Xie., "American sign language recognition using deep learning and computer vision," p. 4896–4899, IEEE, 2018.

[3] C. Dong, M.C. Leu, and Z. Yin, "American sign language alphabet recognition using microsoft kinect," p. 44–52, IEEE, 2015.

[4] S. Ahmed, M. Islam, J. Hassan, M. U. Ahmed, B. J. Ferdosi, S. Saha,M. Shopon et al., "Hand sign to bangla speech: A deep learning invision based system for recognizing hand sign digits and generating bangla speech," in *arXiv preprint arXiv:1901.05613*, 2019.

[5] V. Bheda and D. Radpour, "Using deep convolutional networks for gesture recognition in american sign language," in *arXiv preprintarXiv:1710.06836*, 2017.

[6] Chen, J.K., "Sign language recognition with unsupervised feature learning," in *CS229 Project Final Report; Stanford University, Stanford, CA, USA*, 2011.

[7] Mittal, A.; Kumar, P.; Roy, P.P.; Balasubramanian, R.; Chaudhuri, B.B., "A modified lstm model for continuous sign language recognition using leap motion.," in *IEEE Sens. J.*, p. 7056–7063, 2019.

[8] Aparna, C.; Geetha, M., "Cnn and stacked lstm model for indian sign language recognition," in *Commun. Comput. Inf. Sci*, p. 126–134, 2020.

[9] Jiang, S.; Sun, B.; Wang, L.; Bai, Y.; Li, K.; Fu, Y., "Skeleton aware multi-modal sign language recognition.," in *In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA*, p. 3413–3423, 2021.

[10] JLiao, Y.; Xiong, P.; Min, W.; Min, W.; Lu, J., "Dynamic sign language recognition based on video sequence with blstm-3d residual networks.," in *IEEE Access*, p. 38044–38054, 2019.

[11] Adaloglou, N.; Chatzis, T., "A comprehensive study on deep learning-based methods for sign language recognition.," in *IEEE Trans. Multimed.*, p. 1750–1762, 2022.

[12] Camgoz, N.C., Hadfield, S., Koller, O., Ney, H., Bowden, R., "Neural sign language translation," in *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 18–22, 2018.

[13] "Asl alphabet." `https://www.kaggle.com/datasets/grassknoted/asl-alphabet`.

[14] P. Baldi and P. J. Sadowski, "Understanding dropout," *Advances in neural information processing systems*, vol. 26, 2013.

[15] A. D. Morozov, D. O. Popkov, V. M. Duplyakov, R. F. Mutalova, A. A. Osiptsov, A. L. Vainshtein, E. V. Burnaev, E. V. Shel, and G. V. Paderin, "Data-driven model for hydraulic fracturing design optimization: Focus on building digital database and production forecast," *Journal of Petroleum Science and Engineering*, vol. 194, p. 107504, 2020.

[16] A. Palacios Cuesta, "Hyperparameter optimization for large-scale machine learning," pp. 1–3, 10 2018.

[17] Andonie, Răzvan, "Hyperparameter optimization in learning systems.," pp. 279–291, 2019.

[18] Ian Dewancker, Michael McCourt, Scott Clark, "Bayesian optimization primer,"

[19] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *International conference on machine learning*, pp. 115–123, PMLR, 2013.

[20] Akiba, T., Sano, S., Yanase, T., Ohta, T. and Koyama, M., "Optuna: A next-generation hyperparameter optimization framework," in *In Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery data mining*, pp. 2623–2631, 2019.

[21] Bergstra, J., Bardenet, R., Bengio, Y. and Kégl, B., "Algorithms for hyper-parameter optimization," in *Advances in neural information processing systems*, 2011.

[22] Elsken, T., Metzen, J.H. and Hutter, F., "Neural architecture search: A survey," pp. 1997–2017, 2019.

[23] Javier Arellano-Verdejo, Hugo E. Lazcano-Hernández, "Dataset classification: convolutional neural networks." https://bio-protocol.org/exchange/minidetail?type=30&id=9714110, 2021.

[24] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

[25] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014.

[26] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[28] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[29] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[30] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.