Automated Sign Language Detection using Deep Neural Networks

Smeet Dhakecha, Nithyashree Manohar
dhakecha@usc.edu, nithyash@usc.edu

May 11, 2022

Abstract

Advancements in deep learning algorithms could help the deaf and hard-of-hearing by allowing them to communicate more effectively through computer vision applications. In this project, we explore hand gesture recognition using neural networks. The data set in this work mainly comprises of images. It consists of static images of different hand gestures that corresponds to American Sign Language. In this work, We investigate several deep learning based systems that can correctly identify these hand gestures. The baseline system consists of a dense neural network. We also train the Convolution Neural Network, and ResNet9. We observe that the ResNet9 is our best performing model with a validation accuracy of **99.69%** and test accuracy of **100%** on the test set, and **89%** on the ASL real world test set. The ResNet9 model generalizes well on predicting the confusing hand gestures as well as the real-world images. The application of this model is to help the deaf and hard-of-hearing communicate better in day to day life. This project can be scaled to identify dynamic sign language by capturing images every few milliseconds and using these pre-trained models to identify the sign gestures in real-time.

1 Introduction

Translating Sign language to a text or speech has caught a significant attraction over the last few years. We usually require a machine, or a translator to convert the sign language into human interpretable form, such as text, or speech. Among the many different types of gestures, the American sign language is very well structured, and it has its assigned interpretation, and predefined set of rules for context, and grammar to make the recognition tractable. Most of the deaf people in United States uses American Sign Language(ASL) as a standard choice, that contains over 6,000 gestures[1] for commonly used words, and spelling for communicating obscure words, or nouns. The researches have been actively researching in the automated approaches to detect hand sign, or gestures. With advances in technology and increasing data, deep learning based methods have shown promising results in tackling such problem. Deep learning methods use comprehensive data set to learn set of features that standardize well to unseen data set.

In this project, we aim to solve this problem by employing deep neural network, and Convolutional neural network models. The rest of this report has been organized as follows. Section 2 introduces some of the previous work in this domain. Section 3 describes the data set, preprocessing methods, and how the data has been used for the model training, and the models implemented. Section 4 analyzes the results, and comparison obtained from the model training. Section 5 summarizes the work done in this project, and our concluding remarks.

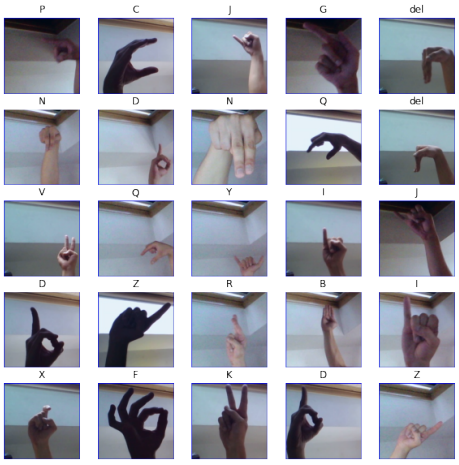


Figure 1: Image showing a few of the hand gestures, and the word class of each of the gestures

2 Literature Review

For the American Sign Language detection, the past approaches that have been implemented uses image processing techniques to manually extract features, locations of fingers, and hand positions to tackle this problem. [1, 2] proposes an HMM system, using a color cameras, to track the upper limb in real time to interpret American sign language. Ohn-bar *et al.* [3] incorporated SVM, with histogram gradient features to achieve state-of-the-art accuracy. A potential problem with employing such problem is that they are prone to the varying scales, location of the sign in the image, and are not generalized when there is a significant change in contrast, or other conditions.

Employing Deep learning model automates the feature extraction process, and also the CNN layers in the model extracts the features at scale, thus ensuring generalizability of the model for real-world applications. This method [4] proposes CNN based approach to classify gestures at image spatial resolutions. We review Convolutional Neural network based approaches, AlexNet, VGG-19, and ResNet models, that have been proven to achieve the state-of-the-art accuracy for the classification task.

3 Approaches and Implementation

3.1 Dataset Usage

In this project, we use the ASL Alphabet dataset[5]. The dataset is a collection of images of alphabets from the American Sign Language, separated in 29 folders, containing images of each of the classes. The training data set contains 87,000 images, of the size are 200×200 pixels. There are 29 classes, of which 26 are for the letters A-Z and 3 classes for SPACE, DELETE and NOTHING. These 3 classes are very helpful in real-time applications, and classification. The test data set contains a mere 29 images. In addition to these 29 images, we used several images captured in real time using different backgrounds to test the model. We also used ASL static image datasets[6] available on Kaggle to test our model. Different datasets yield different test accuracy. During the training phase, we split the train set to make a validation test set. The train and validation set split is 0.8. After the split, our data consists of **69,000** images in the training set and **17,400** images in the validation set. We use the validation set to tune the model hyper-parameters described in the subsequent sections.

3.2 Preprocessing

Building an effective neural network model requires careful consideration of the network architecture as well as the input data format. We have 3000 images for each of the 29 classes, thus we do not perform data augmentation in this project. The other preprocessing operations we performed on the data are listed below.

3.2.1 Resize

We resized our images using the transforms module. This function converts all the images to a specifies size. In this case, we want to resize all the images to 128×128 and hence we pass 32 as an argument to the resize function.

3.2.2 Random Crop

In an image classification task, the test images passed to the model may not have the same dimensions as the train images, and the gesture may not be in the center. To generalize, we perform a random crop operation on the train data set. Size and padding are passed as arguments. The size argument takes an integer that specifies the desired output of size of the random crop. The padding argument takes an integer as input and adds an additional border to the image.

Figure 2: Example image after RandomCrop preprocessing

3.2.3 Data Normalization

Data normalisation is a crucial process that ensures that each input parameter (in this case, pixel) has a uniform data distribution. This allows for faster convergence while training the network. By subtracting the mean from each pixel and dividing the result by the standard deviation, data is normalised. A Gaussian curve centred at zero would be the distribution of such data. Because we need positive pixel counts for picture inputs, we might scale the normalised data in the range [0,1].

3.2.4 Grayscale

From our literature review, we find that the grayscale images achieves more accuracy in the image classification field. Grayscaling is the process of converting an image from other color spaces e.g. RGB, CMYK, HSV, etc. to a single channel gray scale image.

Figure 3: Example image after Grayscaling

3.3 Network Implementations

In this work, we experiment with 3 deep learning approaches, mainly a dense neural network model, a baseline CNN model, and ResNet9.

3.3.1 Dense Neural Network

Model complexity varies with the problem being solved. In a simple case, with a narrow range of both inputs and outputs, and their inherent differences a linear network works well. However, in this case, we required pre-processing, generalization, and a sufficiently large linear network to classify such hand gestures. Also, the number of output classes for this problem is 29 which was difficult for a linear model to handle.

Layer	Parameter	Value
Input	Size	$2 \times H \times W$
Dense1	Input Nodes	$2 \times H \times W$
	Output Nodes	64
ReLU1	Input Nodes	$2 \times H \times W$
	Output Nodes	64
Dense2	Input Nodes	64
	Output Nodes	128
ReLU2	Input Nodes	64
	Output Nodes	128
Dense3	Input Nodes	64
	Output Nodes	29
Softmax	Input Nodes	64
	Output Nodes	29

Table 1: Dense Network Details

3.3.2 Convolutional Neural Network

Our data set consists entirely of images, which necessitates the use of a feed-forward Convolutional neural network with Dense layers because of conventional layers' ability to extract image features at various scales. It also reduces the total number of trainable parameters in the network, thus reducing the inference time at model testing stage.

In our Network, we have incorporated 2 CNN layers with the following network details.

Layer	Parameter	Value
Input	Size	$2 \times H \times W$
Conv1	Input Channel	1
	Output Channel	4
	Stride	1
	Kernel size	3
	Padding	(1, 1)
BatchNorm	Input Channel	4
	Output Channel	4
ReLU	Input Channel	4
	Output Channel	4
MaxPooling2D	Stride	2
Conv2	Input Channel	4
	Output Channel	4
	Stride	1
	Kernel size	3
	Padding	(1, 1)
BatchNorm	Input Channel	4
	Output Channel	4
ReLU	Input Channel	4
	Output Channel	4
MaxPooling2D	Stride	2
Dense	Input Nodes	$2 \times H/4 \times W/4$
	Output Nodes	29

Table 2: CNN Network Details

3.3.3 ResNet9 Network

The ResNet model tackles the issue of diminishing gradients in the deep neural network. The core idea of the Residual connections is to introduce shortcut connections (sometimes also referred to as skip layers) so that the model can take the information from the previous layers to extract features at different scales. There are many variations for ResNet models and we choose ResNet9 [7] in our work.

The figure 4 illustrates the concept of residual connection in a network.

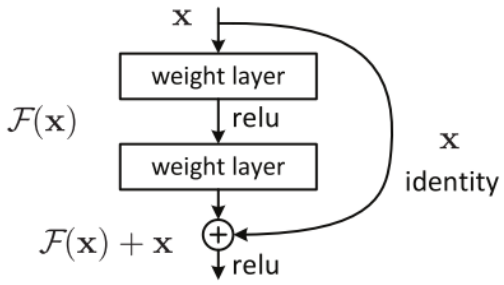


Figure 4: The residual connection in the network

Layers	Parameter	Value
Input	Size	$2 \times H \times W$
Conv1	Input Channel	1
	Output Channel	64
	Stride	1
	Kernel size	3
	Padding	(1, 1)
BatchNorm1	Input Channel	64
	Output Channel	64
ReLU1	Input Channel	64
	Output Channel	64
Conv2	Input Channel	64
	Output Channel	128
	Stride	1
	Kernel size	3
	Padding	(1, 1)
BatchNorm2	Input Channel	128
	Output Channel	128
ReLU2	Input Channel	128
	Output Channel	128
Residual1 (ReLU2 + Conv1)		
Conv3	Input Channel	128
	Output Channel	256
	Stride	1
	Kernel size	3
	Padding	(1, 1)
BatchNorm3	Input Channel	256
	Output Channel	256
ReLU3	Input Channel	256
	Output Channel	256
Conv4	Input Channel	256
	Output Channel	512
	Stride	1
	Kernel size	3
	Padding	(1, 1)
BatchNorm4	Input Channel	512
	Output Channel	512
ReLU4	Input Channel	512
	Output Channel	512
Residual2 (ReLU4 + Conv3)		

Table 3: ResNet9 [7] Network Details

4 Analysis: Comparison of Results, Interpretation

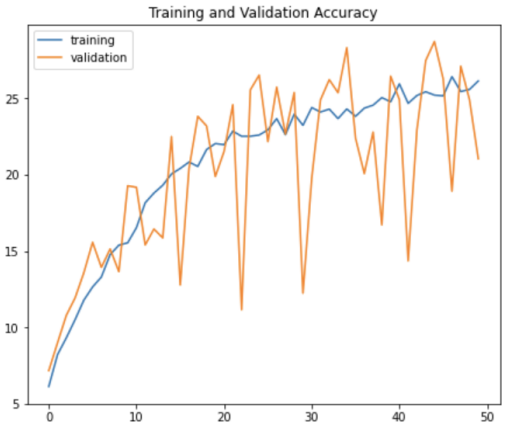
We compare the performance of each of the models, and attach the graphs below. In all of our models, we used Xavier normal initializer for dense layers, and the Kaiming Normal initializer for Convolutional layers. We use Cross Entropy loss for reducing the error, with the weight decay parameter of 0.0001. We utilize the Adam Optimizer [8] for optimizing the loss function. The model has been trained on the same number of images of all the classes. So, we do not investigate any techniques to deal with class imbalance.

4.1 Dense Neural Network

For the dense model, we observe the following variation of loss and accuracy with number of epochs, shown in figure 5, 6. From the graph, we observe that training loss and training accuracy was very stable. We record the training time for this model with 50 epochs to be around 2.5 hours. The validation loss, and accuracy is not stable, and varies too much as the training progresses. A potential reason for such variation is the large number of input nodes.

Despite the stable training, the model only achieved 26.13% training accuracy, and 21.04% validation accuracy.

Figure 5: A Graph showing Loss function variation with epochs in the Dense model



Dense

Figure 6: A Graph showing Accuracy variation with epochs in the Dense model

We observe the test accuracy of 25% with Dense model, and due to such poor performance and unstable training, we do not investigate this model further.

4.2 Convolutional Neural Network

By adding only 2 Convolutional layers, the model performance here improved significantly compared to the dense model. We trained the CNN model for 50 epochs, achieving the **99.82%** Accuracy, but we observed that the validation loss was starting to increase, thus over-fitting the model. So, we obtain the cut-off point beyond which over-fitting was starting which was at 20^{th} Epoch. So, we only train the model for 20 epochs. We record the training time for this model with 20 epochs to be around 2 hours.

We use the validation dataset to tune the model. We conduct the following experiments to tune the model. We have primarily tuned the Batch size, and learning rate parameters.

Table 4: Comparison of the CNN model Performance for various hyper-parameters on train and validation dataset

Model	Hyper-parameter	Dataset	Loss	Accuracy
Baseline CNN Model (Batch size = 32)	Learning rate	Train	0.026	**99.67**
	0.1	Validation	0.537	96.50
	Learning rate	Train	0.054	99.54
	0.01	Validation	0.659	95.77
	Learning rate	Train	**0.014**	99.61
	0.001	Validation	**0.111**	**97.65**
	Learning rate	Train	0.011	99.38
	0.0001	Validation	0.324	96.56
Baseline CNN Model (Batch size = 64)	Learning rate	Train	0.154	98.34
	0.1	Validation	0.892	96.89
	Learning rate	Train	0.099	98.97
	0.01	Validation	0.766	96.28
	Learning rate	Train	0.082	**99.87**
	0.001	Validation	0.791	**97.32**
	Learning rate	Train	**0.043**	99.21
	0.0001	Validation	**0.324**	96.40
Baseline CNN Model (Batch size = 128)	Learning rate	Train	0.261	98.99
	0.1	Validation	0.987	96.20
	Learning rate	Train	0.225	**99.42**
	0.01	Validation	0.681	96.04
	Learning rate	Train	0.142	99.32
	0.001	Validation	**0.643**	**97.55**
	Learning rate	Train	**0.098**	99.09
	0.0001	Validation	0.654	97.33

From the table 4, we observe that as the batch size increases, both the training loss, and the validation loss increases. Also, the accuracy drops slightly as the batch size increases. We observe that the optimal value for the learning rate is 0.001, and the optimal batch size value to be 32, because these parameter values yield the lowest validation loss, and validation accuracy.

We plot the Accuracy and the loss function vs epoch Graphs, with the following parameters:

```
Batch Size = 32
Learning Rate = 0.001
```

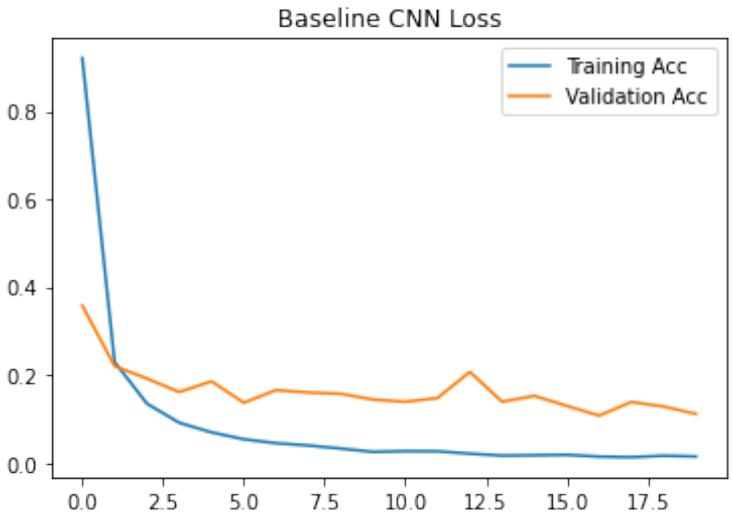


Figure 7: A Graph showing Loss function variation with epochs in the baseline CNN model

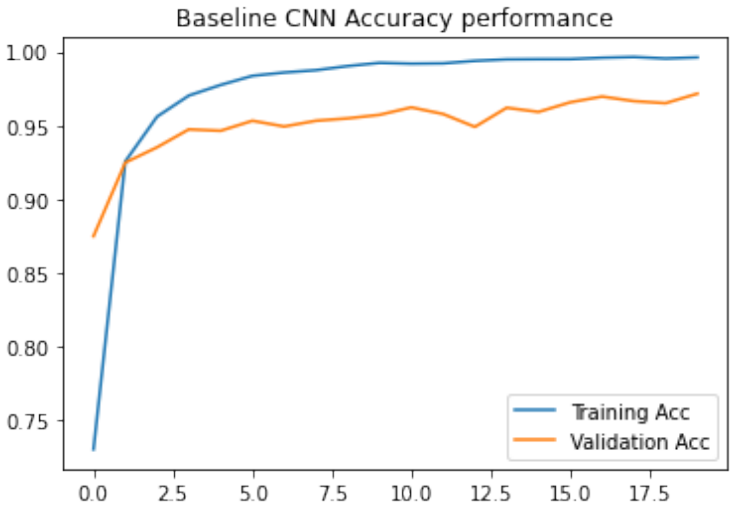


Figure 8: A Graph showing Accuracy variation with epochs in the baseline CNN model

(a) Correct Class:M Predicted Class:M

(b) Correct Class:K Predicted Class:K

(c) Correct Class:H Predicted Class:H

(d) Correct Class:E Predicted Class:E

Figure 9: Some of the predictions of the baseline CNN Model on test data

As you can see in the figure 9 (d), even though there is very poor contrast, the model has been able to predict the correct class. Also, in the figure 9 (c), despite the moving hand gesture, the model is able to give the correct prediction, thus showing generalizability of the network on the unseen images. This CNN model classifies all of the images in the test dataset correctly.

4.3 ResNet9 Network

In the ResNet9, We observe an improvement in the accuracy compared to the baseline CNN model. The one crucial observation that we make is that the model provides the validation accuracy of over 99% within just 3 epochs. In the beginning, we performed training on this network for 50 epochs, and the validation accuracy and the validation loss starts to plateau after 10 epochs, showing no significant improvement. So, here, we train this model only for 10 epochs. We record the training time for this model with 10 epochs to be around 2 hours.

In the ResNet9 CNN network, we conduct the following experiments to tune the hyper-parameters, as shown in the Table 5.

Table 5: Comparison of the ResNet9 model Performance for various hyper-parameters on train and validation dataset

Model	Hyper-parameter	Dataset	Loss	Accuracy
ResNet9 Model (Batch size = 32)	Learning rate	Train	0.0087	99.43
	0.1	Validation	0.0912	99.11
	Learning rate	Train	0.0032	99.69
	0.01	Validation	0.0774	98.90
	Learning rate	Train	**0.0012**	**99.71**
	0.001	Validation	0.0786	**99.69**
	Learning rate	Train	0.0029	99.20
	0.0001	Validation	**0.0043**	99.54
ResNet9 Model (Batch size = 64)	Learning rate	Train	0.0121	98.77
	0.1	Validation	0.0546	98.09
	Learning rate	Train	0.0098	98.60
	0.01	Validation	**0.0231**	98.23
	Learning rate	Train	**0.0071**	98.99
	0.001	Validation	0.0954	**98.65**
	Learning rate	Train	0.0073	**99.04**
	0.0001	Validation	0.0465	98.30
ResNet9 Model (Batch size = 128)	Learning rate	Train	0.0328	98.83
	0.1	Validation	0.0761	98.21
	Learning rate	Train	**0.0213**	99.02
	0.01	Validation	0.0580	**98.92**
	Learning rate	Train	0.0289	**99.14**
	0.001	Validation	0.0997	98.78
	Learning rate	Train	0.0323	98.97
	0.0001	Validation	**0.0133**	98.03

As it can be observed from table 5, we report that loss function value for training, and validation is significantly reduced compared to the loss value in the table 4. Here, the loss function value increases as the batch size is increased. From the table, we obtain the minimum validation loss at the batch size of 32, and the Learning rate value of 0.0001. But highest validation accuracy is observed at the learning rate value of 0.001. So, in selecting the optimal parameter value, we choose validation loss over accuracy.

We plot the Accuracy and the loss function vs epoch Graphs, with the following parameters:

```
Batch Size = 32
Learning Rate = 0.0001
```

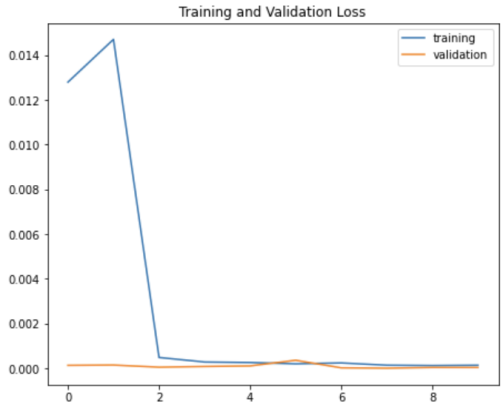


Figure 10: A Graph showing Loss function variation with epochs in the ResNet9 model

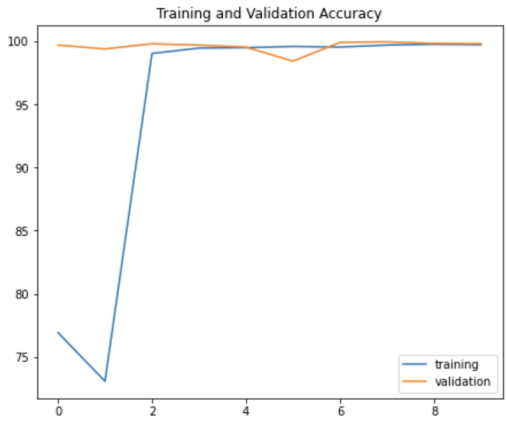


Figure 11: A Graph showing Accuracy variation with epochs in the ResNet9 model

The ResNet9 network gives 100% accuracy on the test dataset that has not been used for training. It also classifies all the images shown in the figure 9 correctly.

We also investigate the model performance on the real world ASL Alphabet Test Dataset [6]. We show the model predictions of the images, in the figure 12.

(a) Correct Class:A
Predicted Class:A

(b) Correct Class:K
Predicted Class:V

(c) Correct Class:X
Predicted Class:X

(d) Correct Class:R
Predicted Class:R

(e) Correct Class:M
Predicted Class:M

(f) Correct Class:L
Predicted Class:L

(g) Correct Class:U
Predicted Class:U

(h) Correct Class:V
Predicted Class:V

(i) Correct Class:DEL
Predicted Class:DEL

Figure 12: ResNet9 Predictions on the real world ASL test dataset

It can be observed in the figure 12 that there are certain images that might seem confusing to classify by visual inspection. Figure 12 (a) & (e) can seem to be interpreted in the same way, because the gesture is almost the same. In the Figure 12 (b) & (h), the hand gesture is very alike, and thus model was able to interpret (b) as class V even though it belongs to class K. We ran the experiment by predicting all the images belong to the class K, and the model was able to classify the images **62%** of the time. Similarly, the model was able to correctly classify all the images belong to the class V **57%** of the time. We observe the **89%** accuracy on this test dataset.

Figure 12 (d) & (g), and (c) & (g) can be confusing to classify as well. Yet, the model is correctly classifying such images almost all the times, ensuring the generalizability of the model. The ResNet9 is very versatile in classifying hand signs from the real-world images, and thus one possible extension of this work is to use video frames to recognize and classify the gestures in real-time.

5 Conclusion

In this work, we aim to automate the process of identifying the hand gestures for sign language. We investigate several approaches for solving this problem. We explore dense neural network based approach, that performs very poorly on both the validation set and test set. We also train a feed-forward CNN model, that provides a significant improvement over dense neural network. By add residual connections in the CNN network, which improves the model performance further. Here, we use the standard ResNet9 architecture, which has been proven to achieve the state-of-the-art architecture. We also experiment with various hyper-parameters to tune the model that gives us the optimal performance. The large training time limits our scope of the number of parameters that can be tuned. In this project, the ResNet9 network classifies images correctly even with lower contrast, hand movement, and yet gives us the desirable performance. So, we report that it can be used to perform real-time sign language detection with the extension of work to include dynamic gesture signs in the training set.

References

[1] Starner, T., Weaver, J. and Pentland, A. (1998) Real-time American sign language recognition using desk and wearable computer based video. IEEE Trans. Pattern Anal. Mach. Intell., 20, 1371–1375.

[2] Ohn-Bar, E. and Trivedi, M.M. (2014) Hand gesture recogni- tion in real time for automotive interfaces: a multimodal vision- based approach and evaluations. IEEE Trans. Intell. Transport. Syst., 15, 2368–2377.

[3] Zaki, M.M. and Shaheen, S.I. (2011) Sign language recognition using a combination of new vision based features. Pattern Recognit. Lett., 32, 572–577.

[4] Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R. and Fei-Fei, L., (2014) Large-Scale Video Classification with Convolutional Neural Networks. IEEE Conf. Computer Vision and Pattern Recognition, Columbus, OH, USA, June 24–27, 2014, pp. 1725–1732. IEEE, New Jersey, USA.

[5] ASL Alphabet Dataset: `https://www.kaggle.com/datasets/grassknoted/asl-alphabet`

[6] ASL Alphabet Testset: `https://www.kaggle.com/datasets/danrasband/asl-alphabet-test`

[7] Kaiming He and Xiangyu Zhang and Shaoqing Ren and Jian Sun. Deep Residual Learning for Image Recognition, 2015. `https://arxiv.org/abs/1512.03385`

[8] Kingma, Diederik, P. and Jimmy, B. Adam: A Method for Stochastic Optimization. doi: 10.48550/ARXIV.1412.6980 `https://arxiv.org/abs/1412.6980`