Computer Vision for Reading American Sign Language Using Convolutional Neural Networks

Project Report

EE 541 - A Computational Introduction to Deep Learning
Prof. Brandon Franzke

Krishna Kamal Adidam (USC ID: 6471530487)
adidam@usc.edu
Ashwani Kumar Pradhan (USC ID: 4900772727)
ashwanik@usc.edu

Abstract

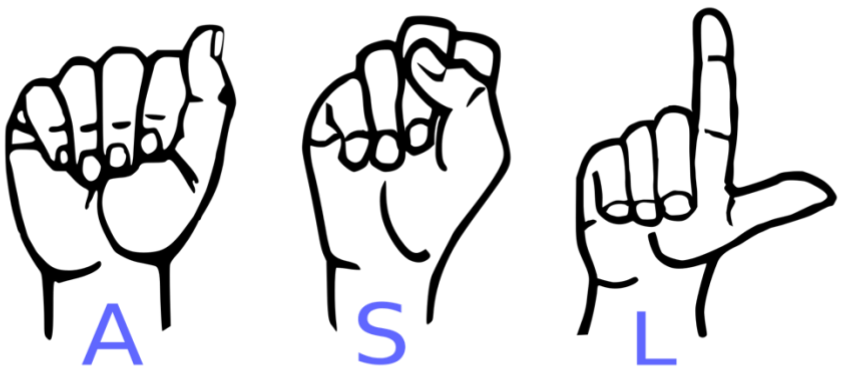
Image Classification is a very vital part of Machine Learning and many algorithms have been developed to do it in an efficient way. Convolution Neural Networks have paved the way to improve it significantly. In this project, to read and recognize the American Sign Language, we have trained various Neural Network Architectures using Convolutional Neural Networks (CNN) by tuning hyperparameters such as kernel window size for CNN, learning rate for optimizers. Also, we trained our model on ResNet50 and ResNet18. Among all these model architectures, we chose the one which gave best accuracy on training set and used that to evaluate the performance on the test set.

Table of Contents

1. Introduction

1.1 Problem Statement and Goals

American Sign Language (ASL) is a natural language, which works as a predominant sign language for deaf communities and hard- of-hearing in the United States of America. With ever-changing technological advancements to make life easy for the ones with a hearing disability, there is still a lot of work to be done. We have chosen this project intending to create a deep learning model with the help of machine learning algorithms, concepts and computer vision applications, which will help deaf communities to communicate effectively and efficiently with the people who are unaware of sign language.



(Fig: 1.1 - Wikipedia)

We propose to create a model to detect American Sign Language using deep neural networks.

1.2 Literature Review

There have been several approaches to convert the sign language into text using the deep learning models. For example, in [1] uses Inception Convolutional Neural Network (CNN) for recognizing spatial features and Recurrent Neural Network (RNN) to train on temporal features. In [2] uses Convolutional Neural Network (CNN) to predicting Sign Language and achieved 95% accuracy. In [3] also uses Convolution Neural Network to identify the sign represented by images and achieved an accuracy of 94.33%. In [8] uses Convolutional Neural Network (CNN) and recorded the weights and model for real-time prediction. In [9] talks about the relevant features of the model, feature extraction and uses Artificial Neural Network (ANN) for classification of signs.
We aim to get on-par accuracy with previously reported approaches and will be trying out different architectures to compare the performance of the models.

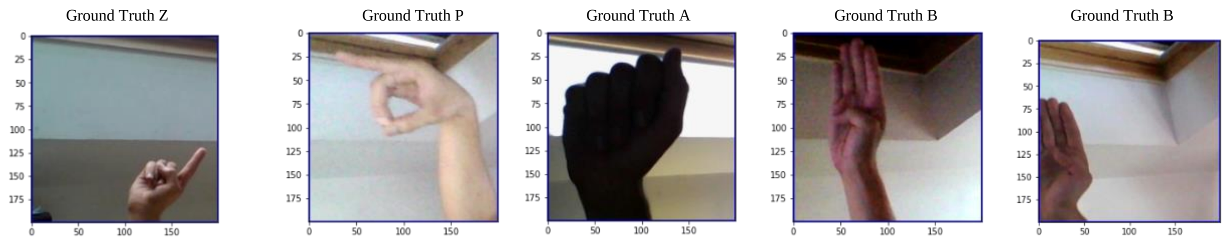
2. Dataset

We are using the dataset provided on Kaggle to train and test our models.

2.1 Dataset Description

On Kaggle we have around 1GB of data, which is a collection of images of alphabets from American Sign Language, separated into 29 folders that represent the various classes. 26 classes consist of 26 alphabets from A-Z and 3 classes have the data for SPACE, DELETE and NOTHING which closely represent the real-world data. The training dataset contains 87,000 images, which are of size 200x200 pixels. The test dataset contains 29 images.

Following are the a few images from the training set:



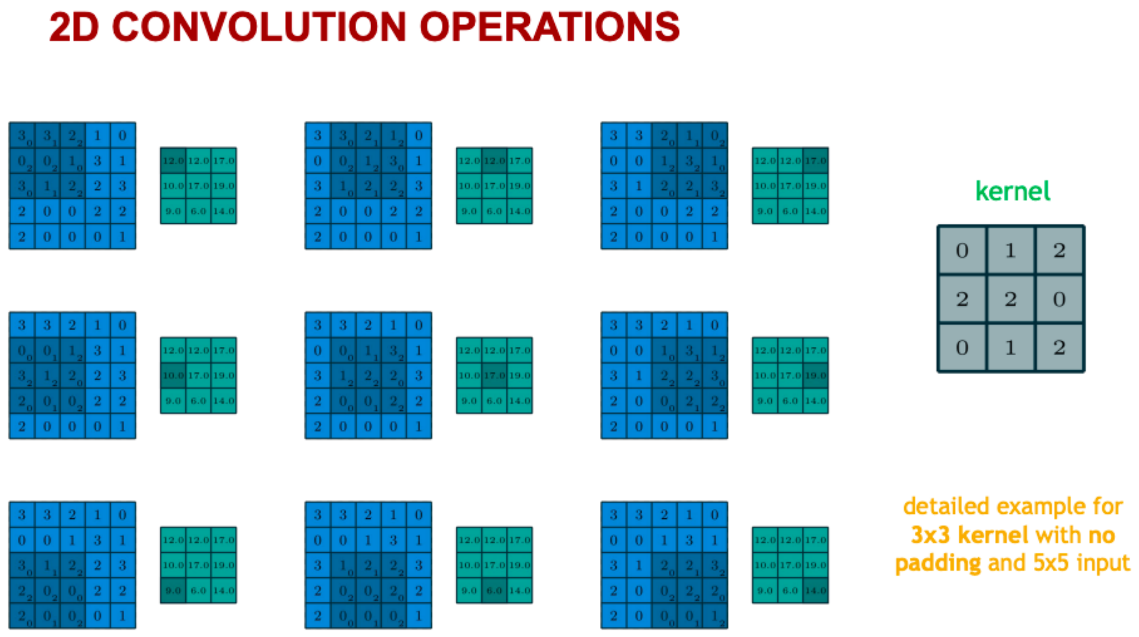
3. Related Work

3.1 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a neural network which has one or more convolutional layers and are mostly used for vision tasks such as image processing, classification, segmentation and for other auto correlated data. We can view convolutions as feature extractors for MLP classifier, in this process this feature extraction is learned.

Convolution is technically sliding a filter over the input, which is explained well with this quote from Dr Prasad Samarakoon: “A convolution can be thought as “looking at a function’s surroundings to make better/accurate predictions of its outcome.” Essentially, we are trying to look at the smaller portion of the image and learn some useful features than looking at the whole image at once and try to learn the features. CNN are very useful when the information is localized.

Convolution Kernel: Each convolutional layer contains a series of filters known as convolutional kernels. The filter is a matrix of integers that are used on a subset of the input pixel values, the same size as the kernel. Each pixel is multiplied by the corresponding value in the kernel, then the result is summed up for a single value for simplicity representing a grid cell, like a pixel, in the output channel/feature map. These are linear transformations; each convolution is a type of affine function.

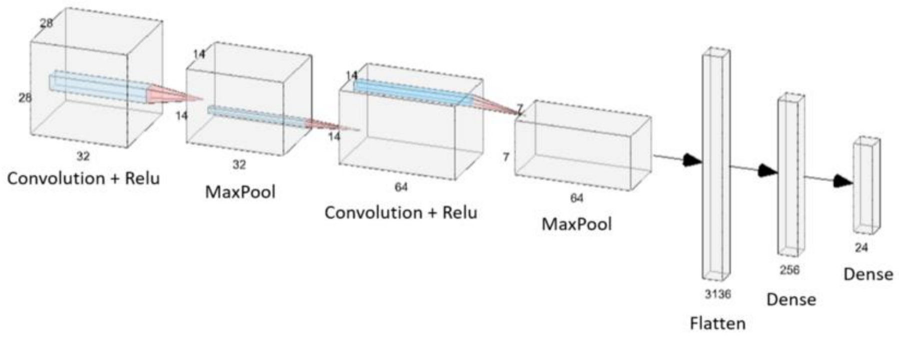

2D CONVOLUTION OPERATIONS
kernel
detailed example for 3x3 kernel with no padding and 5x5 input

(Fig 3.1- EE541: Dumoulin, Vincent and Francesco Visin: A Guide to Convolution Arithmetic for Deep Learning)

The reason to use CNN for our model are as following:

- CNNs are very effective in reducing the number of parameters without losing the quality of the model. As we know, each pixel from images is considered as a feature, so CNNs can be best fit in that case to extract the features.
- CNN retains the 2D spatial form of the images.
- All the layers of a CNN have multiple convolutional filters working and scanning the complete feature matrix and carry out the dimensionality reduction. This enables CNN to be a very apt and fit network for image classifications and processing.

The architecture of example neural network can be seen below:



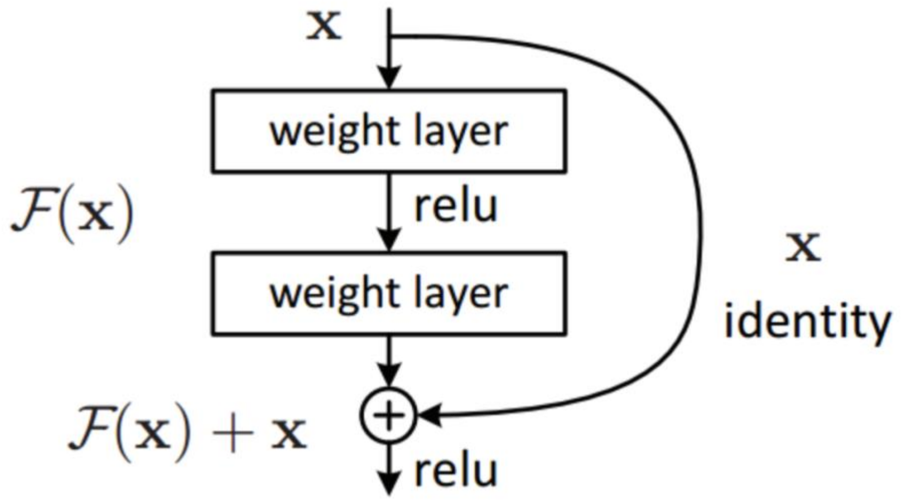
(Fig 3.2 - The architecture of the Convolutional Neural Network, Public Domain)

3.2 Residual Networks (ResNets)

Residual Networks (ResNets) was introduced by Kaiming He, Xiangyu Zhang, Shaoqin Ren, Jian Sun of the Microsoft Research team. This paper explains the degradation problem — When a model gets deeper, after a certain point, the accuracy of the model starts decreasing. This happens because as the model goes deep, it becomes difficult for the layers to propagate information from shallow layers and the information is lost in the process.

From the paper[10]:
"When deeper networks are able to start converging, a degradation problem has been exposed: with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly. Unexpectedly, such degradation is not caused by overfitting, and adding more layers to a suitably deep model leads to higher training error."

The solution provided for this problem was to skip connections. It uses identity networks to directly connect the shallow layers with the deep layers. This way the data can flow easily between two layers and in case any specific layer is affecting the performance of the model, then that layer will be skipped. Thus, being called as skip connection.



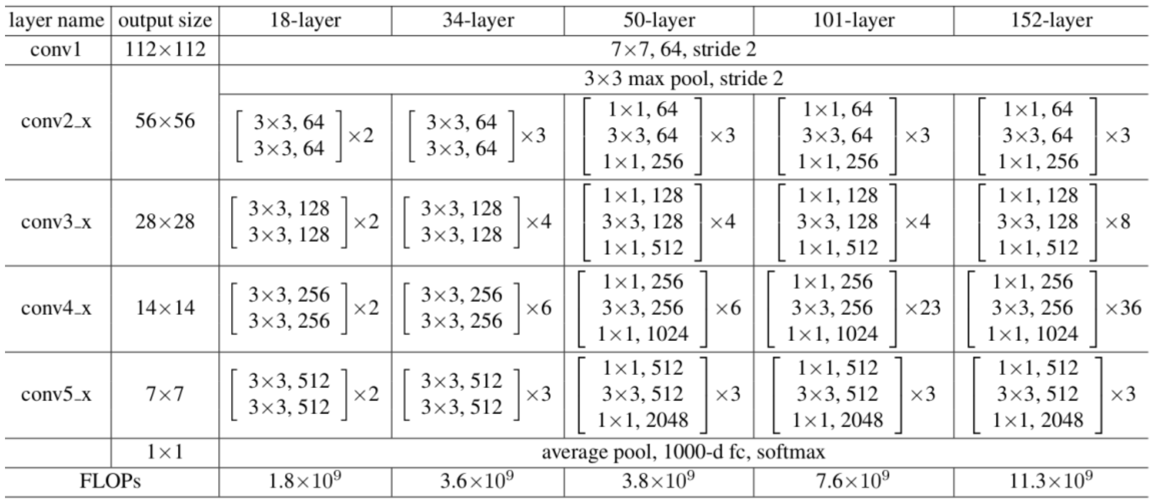
Skip Connect or Shortcut Connection, Image Credits to the authors of original ResNet paper(Source)

(Fig 3.3 – From the original ResNet paper [10] arXiv:1512.03385v1)

Residual Networks – y = f(x) + x

Here +x is for the skip connection and f(x) denotes the residual mapping to be learned. f(x)+x is obtained by elementwise addition of skip connection.

In our project, we are also using ResNet50 and ResNet18 to train our models. The general architecture for same is shown below:



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\left[\begin{array}{c}3\times3, 64\\3\times3, 64\end{array}\right]\times2$	$\left[\begin{array}{c}3\times3, 64\\3\times3, 64\end{array}\right]\times3$	$\left[\begin{array}{c}1\times1, 64\\3\times3, 64\\1\times1, 256\end{array}\right]\times3$	$\left[\begin{array}{c}1\times1, 64\\3\times3, 64\\1\times1, 256\end{array}\right]\times3$	$\left[\begin{array}{c}1\times1, 64\\3\times3, 64\\1\times1, 256\end{array}\right]\times3$
conv3_x	28×28	$\left[\begin{array}{c}3\times3, 128\\3\times3, 128\end{array}\right]\times2$	$\left[\begin{array}{c}3\times3, 128\\3\times3, 128\end{array}\right]\times4$	$\left[\begin{array}{c}1\times1, 128\\3\times3, 128\\1\times1, 512\end{array}\right]\times4$	$\left[\begin{array}{c}1\times1, 128\\3\times3, 128\\1\times1, 512\end{array}\right]\times4$	$\left[\begin{array}{c}1\times1, 128\\3\times3, 128\\1\times1, 512\end{array}\right]\times8$
conv4_x	14×14	$\left[\begin{array}{c}3\times3, 256\\3\times3, 256\end{array}\right]\times2$	$\left[\begin{array}{c}3\times3, 256\\3\times3, 256\end{array}\right]\times6$	$\left[\begin{array}{c}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{array}\right]\times6$	$\left[\begin{array}{c}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{array}\right]\times23$	$\left[\begin{array}{c}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{array}\right]\times36$
conv5_x	7×7	$\left[\begin{array}{c}3\times3, 512\\3\times3, 512\end{array}\right]\times2$	$\left[\begin{array}{c}3\times3, 512\\3\times3, 512\end{array}\right]\times3$	$\left[\begin{array}{c}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{array}\right]\times3$	$\left[\begin{array}{c}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{array}\right]\times3$	$\left[\begin{array}{c}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{array}\right]\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

(Fig 3.4 – Transfer learning model ResNet- 18,34,50,101, 152. source)

4. Model Architecture

The idea is to start with a simple Convolutional Layer (Conv2d) in a CNN and check its performance on the train set. Basing on this, we tune the model architecture by increasing the number of Convolutional layers, adding dropouts, batch Normalization and changing the Kernel window size appropriately. More details on the various architectures implemented for training are mentioned in section 4.3.

4.1 Loss Function

We are using Multiclass Cross Entropy. The cross entropy is defined as:

$$LCE = -\sum_{i=1}^{n} ti \log(pi), \qquad for\ n\ classes$$

Where t_i is the true labels and p_i is the Softmax probability output for the i^{th} class.

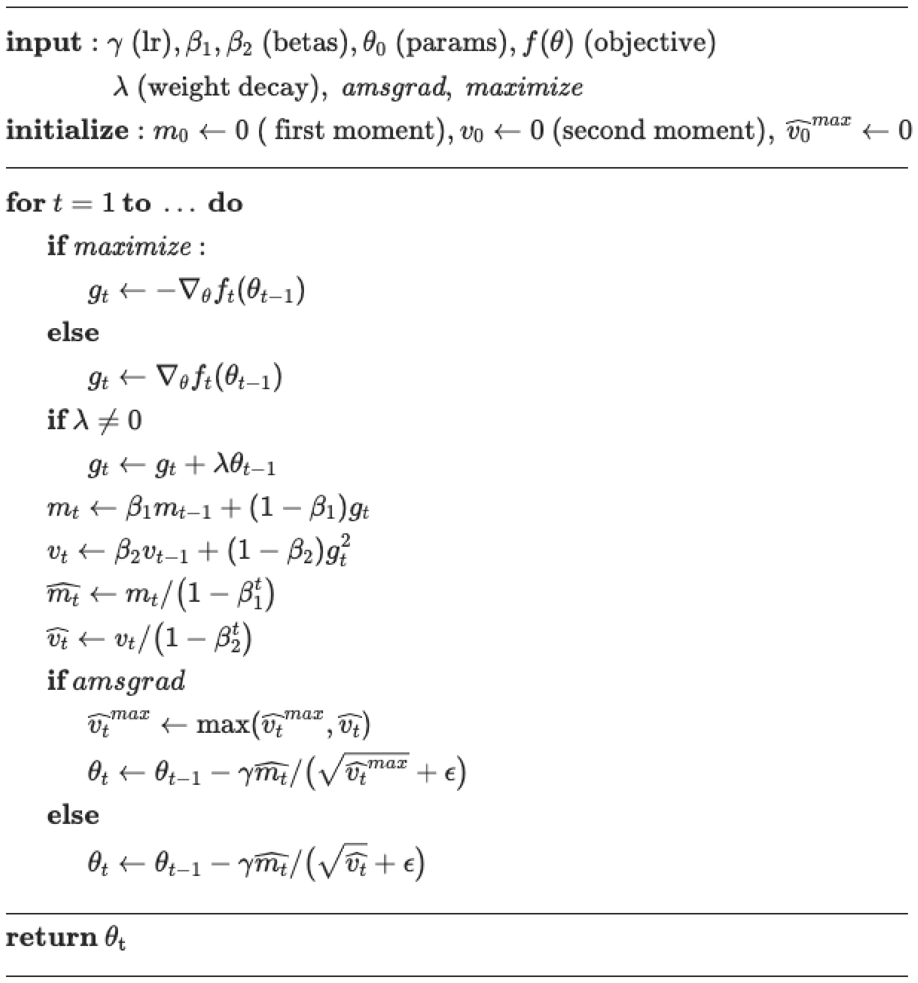
For the two-class problem, we can use Binary Cross Entropy. But we have 29 classes, it is a multi-class classification. So, we use Multiclass Cross Entropy.

4.2 Optimizer

Adam Optimizer: Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data.
Adam is basically combining the advantage of two other extensions of stochastic gradient – Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp)

Adaptive Gradient Algorithm (AdaGrad) - which maintains a per-parameter learning rate that improves performance on problems with sparse gradients
Root Mean Square Propagation (RMSProp) - which also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight. This means the algorithm does well on online and non-stationary problems.

Following is the Adam algorithm:



```
input : $\gamma$ (lr), $\beta_1, \beta_2$ (betas), $\theta_0$ (params), $f(\theta)$ (objective)
        $\lambda$ (weight decay), $amsgrad$, $maximize$
initialize : $m_0 \leftarrow 0$ ( first moment), $v_0 \leftarrow 0$ (second moment), $\widehat{v_0}^{max} \leftarrow 0$

for $t = 1$ to ... do
    if $maximize$ :
        $g_t \leftarrow -\nabla_\theta f_t(\theta_{t-1})$
    else
        $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
    if $\lambda \neq 0$
        $g_t \leftarrow g_t + \lambda\theta_{t-1}$
    $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$
    $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
    $\widehat{m_t} \leftarrow m_t / \big(1-\beta_1^t\big)$
    $\widehat{v_t} \leftarrow v_t / \big(1-\beta_2^t\big)$
    if $amsgrad$
        $\widehat{v_t}^{max} \leftarrow \max(\widehat{v_t}^{max}, \widehat{v_t})$
        $\theta_t \leftarrow \theta_{t-1} - \gamma\widehat{m_t} / \big(\sqrt{\widehat{v_t}^{max}} + \epsilon\big)$
    else
        $\theta_t \leftarrow \theta_{t-1} - \gamma\widehat{m_t} / \big(\sqrt{\widehat{v_t}} + \epsilon\big)$

return $\theta_t$
```

(Fig 4.1 – Adam Implementation Algorithm(source))

4.3 Performance Analysis

We have performed hyperparameter tuning on various CNN architectures to get the best performing model.

Following table shows the different model architectures implemented in this process along with the accuracy scores:

Number of Convolutional Layers	Kernel Window Size	Learning Rate	Loss Function	Optimizer	Regularization Coefficient	Accuracy on train set (%)
1	5 x 5	0.01	Cross Entropy Loss	Adam	0.0001	43.08
2	5 x 5	0.01	Cross Entropy Loss	Adam	0.0001	52.172
3	5 x 5	Schedule-0.01 to 0.001	Cross Entropy Loss	Adam	0.0001	71.827
3	3 x 3	Schedule-0.01 to 0.001	Cross Entropy Loss	Adam	0.0001	73.258

(Table 1: Various CNN models implemented by us)

Following are the ResNet architectures and their related parameters used by us to train the models:

Model Architecture	Loss Function	Optimizer	Regularization Coefficient	Learning Rate	Accuracy on train set (%)
ResNet50	Cross Entropy Loss	Adam	0.0001	Schedule-0.01 to 0.001	98.09
ResNets18	Cross Entropy Loss	Adam	0.0001	Schedule-0.01 to 0.001	99.91

(Table 2: ResNet Architectures)

For the case of 3 convolutional layers, we reduced the learning rate after number of epochs reach half the total number of epochs to get better performance on the fitted model.

Clearly, with help of correct set of parameters, as we go deep, with increase in the number of convolutional layers, Accuracy increases. As we go deep, feature learning is more focused and appropriate set of features are learned by the model. Hence, the accuracy increases. But after a point, it saturates, Hence, we shift to ResNets which make use of skip connections.

5. Challenges

- **Hardware Limitation:** The computational complexity of CNN is very high. We have got each picture with dimension of 200x200 with 3 channels, so if we apply MxN filter then each pixel needs O(MxN) computations. Overall, the 2D convolution would have an approximate complexity of (200x200x3xMxN). Even if we take small values of M and N such that M=N=2, then also we have 0.48M calculations. So, these models were occupying a good amount of GPU memory and we had to leverage many online applications to use the GPUs available on cloud.

- Continuous monitoring of training process of the models and making the hyperparameter changes to obtain the best possible result in timely manner.

6. Future Work

- Designing a model to interpret the motion involved signs; in the real world, we have motion involved for letters 'J 'and 'Z', whereas the Kaggle dataset does not have the motion involved for these letters.

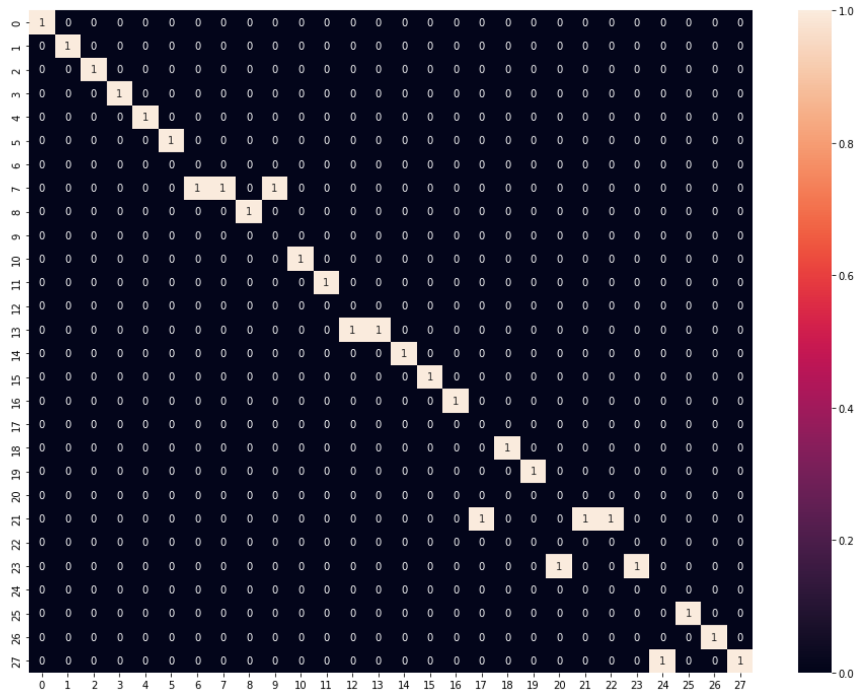
- Designing a model to interpret the sign language in a real time by generating the real time data, feeding it through the model and identifying the words or numbers through that.

7. Applications

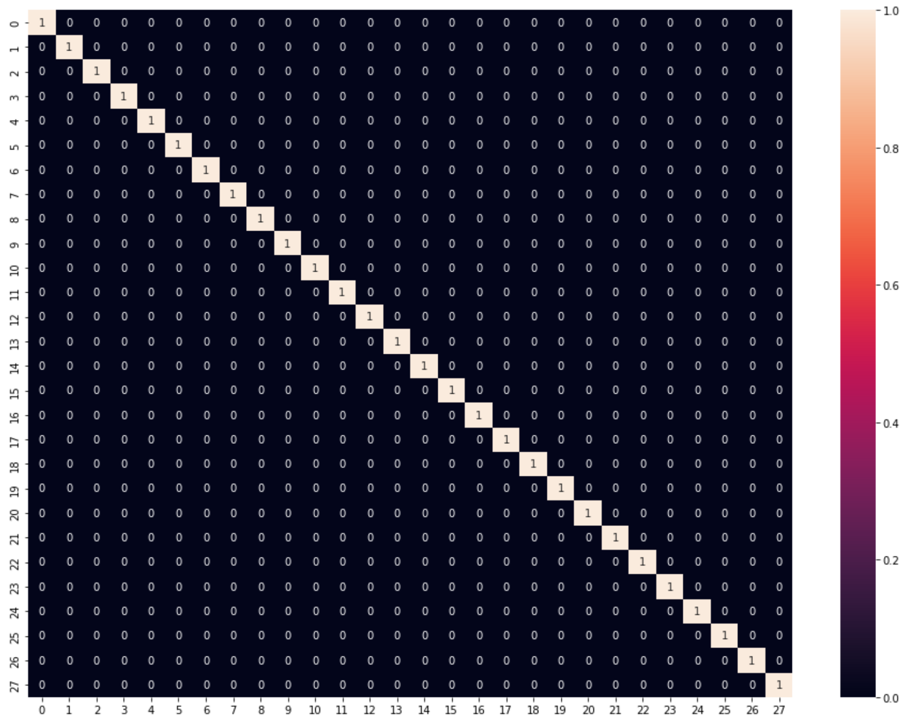
CNNs have changed the game for many computer vision tasks and models built here can be used to understand the static motions for sign language with very high accuracy.

8. Results and Observations

Using the 3 convolutional layers model we have implemented, the accuracy on the test set is 75 % and the corresponding mean F1 score is 0.6785. The heatmap for the same is shown



Using the ResNet18 Model, after training it on the train set, the accuracy on the test set is 100 % and the corresponding mean F1 score is 1.



We expect the test accuracy to be this high because the test set size is very low and after verifying, we can affirm that our model fits good on training data and likewise the performance is same on the test set as well.

9. Conclusion and Discussion

The aim of this project was to build a model to for the task of multiclass classification of American Sign Language. In this project, we compared the different CNN models for hand gesture recognition for ASL. After performing model selection using various CNN architectures and different kernel window sizes, we found that CNN with 3 convolutional layers and a kernel width of (3x3) was the best performing model with accuracy of 75% and mean F1 score of 0.6785 on the test data. Then we implemented the state-of-the-art ResNet18 and ResNet50 models and got the best performing model on train set as ResNet18 model. So, we predicted the test set labels using the ResNet18 model and got an accuracy of 100% with mean F1 score of 1**.**

The GitHub repository link to the code and setup instructions can be found here

References

[1] K. Bantupalli and Y. Xie. American sign language recognition using deep learning and computer vision. In 2018 IEEE International Conference on Big Data (Big Data), pages 4896–4899. IEEE, 2018.

[2] L. Y. Bin, G. Y. Huann, and L. K. Yun. Study of convolutional neural network in recognizing static american sign language. In 2019 IEEE International Conference on Signal and Image Processing Applications (ICSIPA), pages 41–45. IEEE, 2019.

[3] Rawini Dias. American Sign Language Hand Gesture Recognition.
https://towardsdatascience.com/american-sign-language-hand-gesture-recognition-f1c4468fb177

[4] Justin Chen. CS231A Course Project Final Report Sign Language Recognition with Unsupervised Feature Learning, Stnaford University

[5] Chandhini Grandhi, Sean Liu, Divyank Rahoria. American Sign Language Recognition using Deep Learning

[6] vaishshells. Sign Language Recognition for Computer Vision Enthusiasts.
https://www.analyticsvidhya.com/blog/2021/06/sign-language-recognition-for-computer-vision-enthusiasts/

[7] Christopher Thomas. An introduction to Convolutional Neural Networks.
https://towardsdatascience.com/an-introduction-to-convolutional-neural-networks-eb0b60b58fd7

[8] M. Taskiran, M. Killioglu, and N. Kahraman. A real-time system for recognition of american sign language by using deep learning. In 2018 41st International Conference on Telecommunications and Signal Processing (TSP), pages 1–5. IEEE, 2018

[9] A. Thongtawee, O. Pinsanoh, and Y. Kitjaidure. A novel feature extraction for american sign language recognition using webcam. In 2018 11th Biomedical Engineering International Conference (BMEiCON), pages 1–5. IEEE, 2018.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs.CV] (or arXiv:1512.03385v1 [cs.CV] for this version)
https://doi.org/10.48550/arXiv.1512.03385

[11] PyTorch Documentation