American Sign Language Reader using Deep Learning Algorithms

Course Project Report
for
EE-541 | A Computational Introduction to Deep Learning
Spring - 2022

Prepared By
Group 21
Rudrax Dave - USC ID: 6954322559
(rddave@usc.edu)
Rajath Ramegowda – USC ID: 3026555351
(rramegow@usc.edu)

Guided By:

Brandon Franzke

University of Southern California- Viterbi School of Engineering

Special Thanks to:
TAs: Tianchen Yuan, Ganning Zhao | CPs: Aditya Anulekh Mantri, Sarthak Maharana

Abstract

Many differently abled - deaf people in America use American Sign Language (ASL) as their primary language. Gestures are nonverbally communicated messages that are recognized by eyesight. Sign language is the nonverbal communication of the deaf and dumb. The language is as well-defined as spoken languages, and it employs hand gestures, facial movements, and bodily postures. However, we have observed that there are numerous obstacles to accomplishing this activity in person: it necessitates labor, and employed personnel are not always monitored and organized as they should be: There is a need for a dependable system that assists differently abled people in learning, communicating, and creating their own language with accuracy, as well as video to words conversion.

Table of Contents

Table of Figures:

Introduction

This project allows you to put your abilities learned in Introduction to Deep Learning in Python and Image Processing with Pytorch in Python into practice, such as creating convolutional neural networks to categorize photos.

Computer vision systems that convert sign language to spoken language have made significant development in recent years. Complex neural network topologies are frequently used in this technology to identify tiny patterns in live video. However, understanding how to construct a translation system is the first step. In this notebook, we'll learn to categorize photos of American Sign Language (ASL) letters using a convolutional neural network. We will train the network and assess its performance after loading, inspecting, and preparing the data.

Problem Statement and Goals

The goal of this problem is to build a deep learning model to read the American Sign Language. We used the dataset found on Kaggle to build your model and predict the best Model to be Used out of the compared.

Literature Review

We know from previous research that sign language is one of the oldest and most natural forms of communication, but because most people do not know sign language and interpreters are hard to come by, we developed a real-time method for fingerspelling-based American sign language using neural networks. The hand is first sent through a filter in our technique, and then the hand is passed through a classifier, which predicts the class of hand movements.

In recent years, there has been study on the identification of Sign Language. We discovered the essential phases in hand gesture detection using a literature survey: - Data pretreatment, MODEL selection, Classification, Analytics, Video – to - Letters

Approach and Implementation

We were well aware that, in actuality, we should choose a model, parameter, or approach depending on the validation results. We Observed We propose to use two Deep Learning Neural Network models: CNN - Convolutional Neural Networks and RESNET to create a model to generate a Classification prediction for the Alphabets given by the American Sign Language. We also plan to work with different network models and finalize the best suited and highest accuracy model.

We plan to Experiment with different architectures and hyperparameters. First, implement the CNN architecture described in the Models of CNN and apply our training data to generate the classification of 26 Letters. Our goal is to map images from a particular domain to a Letter that it means. We will be finetuning (transfer learning) existing models that perform classification on RGB images. We will Look at the levels of RGB images in Particular Centroidal Pixels and We can Classify from the Centre Which Shape is Defined. We are Planning to Use RESNET - Residual Neural Networks to be able to argue and analyze the better Accuracy between all of the Networks.

Workflow

Our Approach was

1. Preprocessing and loading the dataset
2. Literature Survey and References
3. Analysis, Standardization and Normalization
4. Using Neural Networks and Defining Networks which will support our Dataset.
5. Implementation of Different Techniques – Models
6. Analyzing the performance and choosing the best
7. Conclusion

Dataset Usage

Dataset: https://www.kaggle.com/datasets/grassknoted/asl-alphabet

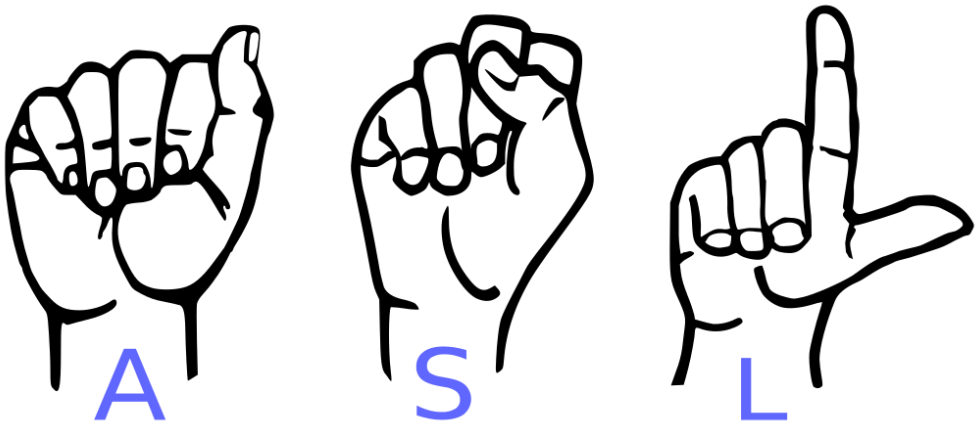

Figure 1. ASL Kaggle DATASET Cover

Overview of dataset usage:

The Dataset is 1.11 GB in Size. The images in the dataset are manually captured and not computer-generated. The dataset linked above contains images from 29 classes (26 alphabets, SPACE, DELETE, and NOTHING). Each class contains 3000 images in the training set and each image is a 200 x 200 RGB image. The training data set contains 87,000 images, 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, to encourage the use of real-world test images.>the test set is very small.

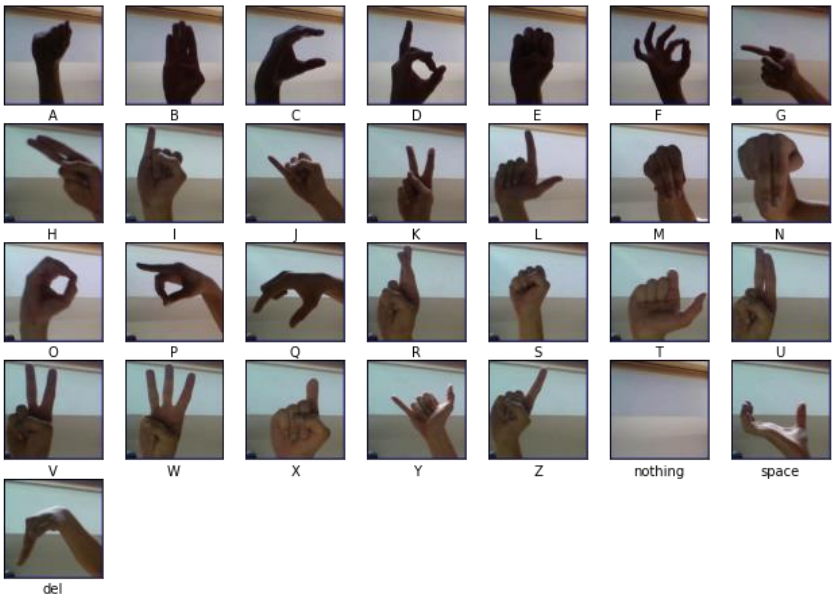

Figure 2: How the DataSet Looks {Code}

Convolution Neural Network:

In contrast to normal Neural Networks, the neurons in CNN layers are organized in three dimensions: width, height, and depth. Instead of being entirely linked, the neurons in a layer will only be connected to a tiny portion of the layer (window size) before to it. The final output layer would also have dimensions (number of classes).

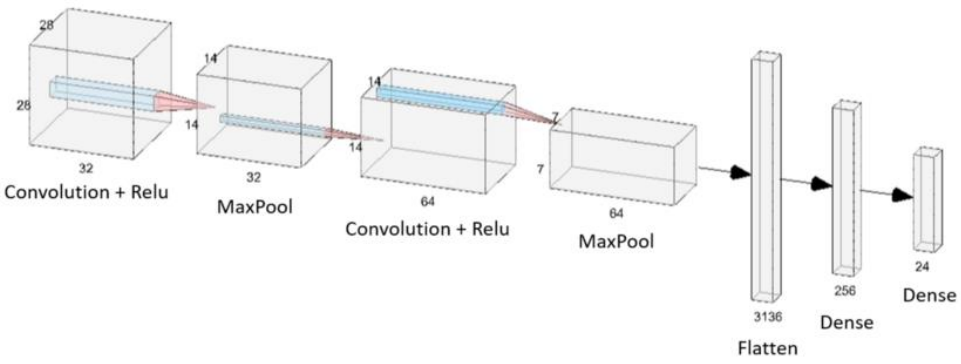

Figure 3:Convolutional Neural Network Explained {Internet}

Convolution Layer:

In convolution layer we take a small window size of length 3 X 3 or 5 X 5 that extends to the depth of the input matrix. During every iteration we slid the window by stride size 1 or 2, and compute the dot product of filter entries and input values at a given position. As we continue this process well create a 2-Dimensional activation matrix that gives the response of that matrix at

every spatial position. That is, the network will learn filters
that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color 2.

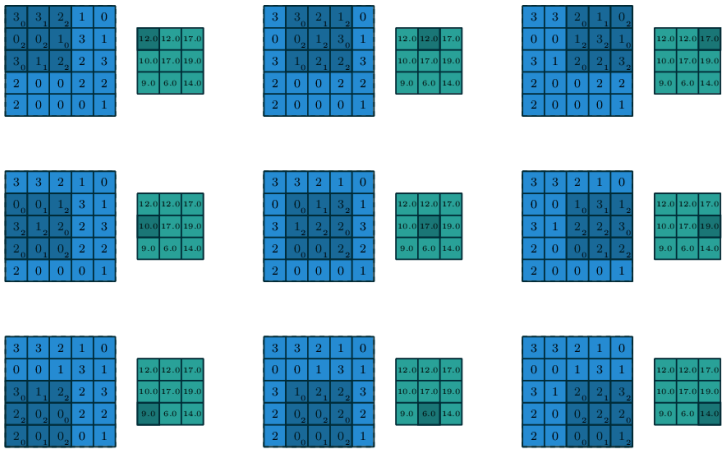

Figure 4: How CNN Layers Work{Source Lecture Slides}

The input picture has resolution of 32 x 32 pixels. It is first processed in the first convolutional layer using 16,32,64,etc filter weights (3x3 pixels each). This will result in a next layer of pixel image, one for each Filter-weights.

Pooling Layer:

We use pooling layer to decrease the size of activation matrix and ultimately reduce the learnable parameters. We Use

Max Pooling:

In max pooling we take a window size of size 2*2, and only take the maximum of 4 values. Well lid this window and continue this process, so well finally get a activation matrix half of its original Size. The pictures are down sampled using max pooling of 2x2 i.e., we keep the highest value in the 2x2 square of array.

Activation Function:

In each of the layers, we utilized ReLu (Rectified Linear Unit) (convolutional as well as fully connected neurons). For each input pixel, ReLu calculates max(x,0). This gives the formula additional nonlinearity and makes it easier to learn more intricate features. It aids in the removal of the vanishing gradient problem as well as the acceleration of training by lowering calculation

time. This minimizes the number of parameters, lowering the cost of computation and reducing overfitting.

Dropout Layers:

Overfitting is an issue in which the network's weights are so tuned to the training examples that it fails to function effectively when given fresh instances after training. By setting the activations in that layer to zero, this layer "drops out" a random collection of activations. Even if part of the activations are left out, the network should be able to produce the appropriate categorization or output for a given case.

Optimizer:

We have used Adam optimizer for updating the model in response to the output of the loss function. Adam combines the advantages of two extensions of two stochastic gradient descent algorithms namely adaptive gradient algorithm (ADA GRAD) and root mean square propagation (RMSProp)

SGD: stochastic gradient descent Optimizer:

The iterative method of stochastic gradient descent (commonly abbreviated SGD) for maximizing an objective function with sufficient smoothness criteria (e.g. differentiable or subdifferentiable).

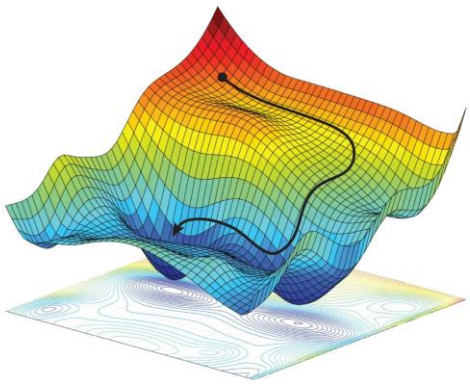

Figure 5: Optimizer {by SGD Documentation}

Loss Function:

To make the network usable, we must first train it so that the weights are meaningful, non-random values. As previously said, we'll utilize the network to produce predictions, then use the

loss function to compare the forecasts to the ground truth.

We may use the following loss functions in PyTorch: For a multi-class classification issue, use nn.CrossEntropyLoss().

Cross entropy Loss:

It is useful when training a classification problem with C classes. If provided, the optional argument weight should be a 1D Tensor assigning weight to each of the classes. This is particularly useful when you have an unbalanced training set.

$$\ell(x,y)=L=\{l1,\ldots,lN\}\top,ln=-wynlog\sum c=1Cexp(xn,c)exp(xn,yn)\cdot 1$$

Fully Connected Layer:

In convolution layer neurons are connected only to a local region, while in a fully connected region, well connect the all the inputs to neurons.

Final Output Layer:

After collecting data from the fully connected layer, link them to the last layer of neurons, which will be the output classes we need to estimate the likelihood of each picture falling into different classes. The Connected Layer's output serves as an input for the final layer, which will contain the same number of neurons as the number of classes we're categorizing (alphabets + blank symbol).

CNN Models:

1 Layer CNN:

It is a Sequentially defined model in Torch by the following:

```
self.NN = nn.Sequential(
        nn.Conv2d(3,16,kernel_size=5,stride=1,padding=2),
        nn.BatchNorm2d(16),
        nn.ReLU(),
        nn.MaxPool2d((2,2),stride=2)
    )
```

Convolution Layer= Conv2d with input picture has
resolution of 32 x 32, but we take 3 colors RGB as the input and 16 filters, with kernel of 5 X 5, stride = 1, and padding = 2.
Batch Normalization = BatchNorm2d(16) – Normalize the output batch to 16.
ReLU= Activation Function
Pooling = MaxPool2d
We also, flatten the Output using Linear (16*100*100,100), Use Dropout Layer = Dropout (0.2), BatchNorm1d(100), Dropout(0.2), Linear(100,29), Softmax(dim=1), as the final Output Layer Activation.
Epochs = 5
Batch Size = 200
Learning Rate = 0.01
With and Without weight_decay = 0.0001
Optimization and Weight Decay PyTorch also computes derivatives for us using automatic

2 Layer CNN:

With 3 - 16 -32 Layers, Similar to 1 CNN but one more layer.

3 Layer CNN:

By trying and testing different model, we learned that the 3 Layer Models actually generalize the data better and gives a suitable output if we fine tune it in a certain manner.
Layer 1
nn.Conv2d(3,64,kernel_size=3,stride=1,padding='same'),
nn.BatchNorm2d(64),
nn.ReLU(),
nn.MaxPool2d((2,2),stride=2),
Layer 2
nn.Conv2d(64,128,kernel_size=3,stride=1,padding='same'),
nn.BatchNorm2d(128),
nn.ReLU(),

nn.MaxPool2d((2,2),stride=2),

Layer 3

nn.Conv2d(128,256,kernel_size=3,stride=1,padding='same'),

nn.BatchNorm2d(256),

nn.ReLU(),

nn.MaxPool2d((2,2),stride=2)

It takes 64 Filters directly and keeps on Doubling down the neurons to increase and train it extensively.

We tried it in 3 x 3 kernel and 5 x 5 kernels and we found great results.

5 Layer CNN:

We found the pattern that increasing layers might improve accuracy: that the 5 Layer Models actually can help, so we trained it -> it gave better results overall but reduced the results from 3 Layer version.

RESNET:

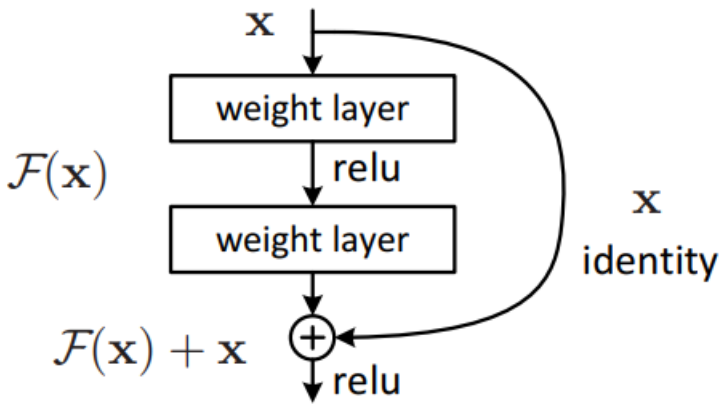

Figure 6: RESNET {Internet}

Short Form of Residual Network. Additional layers are added to a DNN to improve accuracy and performance and are useful in solving complex problems. The intuition was that these layers would progressively learn the features.

model_resnet =

torchvision.models.resnet50(pretrained=True)

We Used RESNET50: That is 50 layers deep. So, we Used the Given built in with our parameter Tunings and Tweaking and Found out that RESNET works the best.

In that we tried these Models:

Model 1

ResNet-50 CNN

Cost Function = CrossEntropyLoss

Optimizer = Adam

Learning Rate = 0.001

Model 2

ResNet-50 CNN

Cost Function = CrossEntropyLoss

Optimizer = SGD

Learning Rate = 0.001

Training and Testing:

We can see that the Dataset is not regularly distributed:

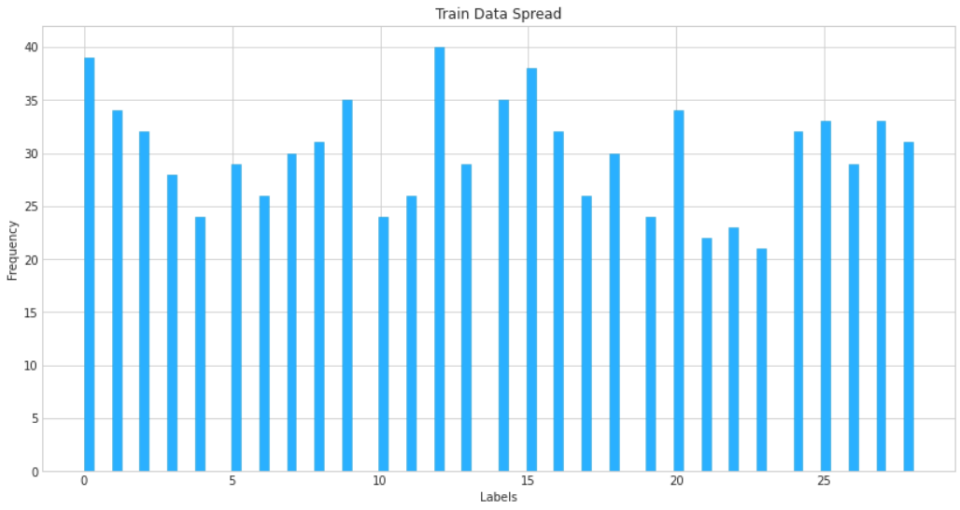

Figure 7: Training Data Distribution {Code}

We found out that there are many different Models to
Consider and, we can Fine tune them to make them predict and read them flawlessly.

Challenges Faced:

There were many challenges faced by us during the project. The very first issue we faced was of dataset.

1. Dataset had Image Files, so it was hard to Load them in Open Notebook Platforms.
2. GPU Selection for Time Reduction – We found that even in Google Colab GPU will take us 1 Hr to train the dataset by a MODEL- So we Choose Kaggle and Their Open GPU Services.
3. More issues were faced relating to the accuracy of the model we trained in earlier phases which we eventually improved by increasing the input Parameters and also by improving the Optimizers and Way our code worked.

Results:

These Accuracies are subject to change on the GPU, Devices used, and Time Taken.
We have achieved an Accuracy: 57.89 in our Single Layer model using only layer 1 of our algorithm, Accuracy60.89 with Using Weight Decay regularize in the same Model with kernel = 5.

Taking Model 2: 2 Layers CNN we got Accuracy: 55.880459770114946.

Using 3 Layer CNNs with proper Parameters, changed the low accuracy values and shoed us the best accuracies possible.
3 Layer CNNs: Accuracy: 91.20 with 3x3 kernel
And Accuracy: 91.77 with 5x5 kernel. -> which can be proven further as the receptive field of the 5x5 kernel can be covered by two 3x3 kernels.

Trying out the 5 Layer CNNs: Accuracy: 91.40 which was lower than 3 layered.
We found Accuracies which are a better accuracy then most of the current research papers on American sign language. You can Look at the table below for results.

RESNET: Model 1 gave us: Accuracy: 96.24942528735633 Model 2 gave us Accuracy: 98.226 with SGD Optimizer.

So, we think that if we want to use predetermined Models, with finetuning, we can try RESNET for Reading the Sign Recognition, but we can also say that 3 Layer CNN works the best as well to predict for 97.22% Accuracy.

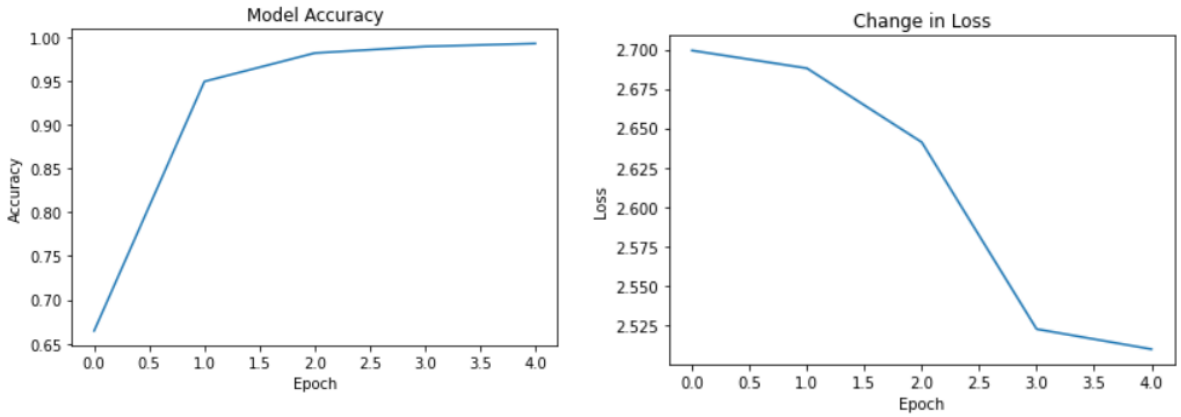

Figure 8: Model Accuracy Epoch Based 3CNN and Losses Visualized

Model selection based on accuracy among CNN models(based on kernels' size)

Hyperparameters selected are as follows:

- Cost Function: CrossEntropyLoss
- Optimizer: Adam
- Learning Rate: 0.001
- Number of Epoch to find accuracy: 5

Model (CNN convolution layer)	Loss		Accuracy	
	Kernel=3	Kernel=5	Kernel=3	Kernel=5
1 layer (Baseline model)	2.91	2.76	60.89	57.89
2 Layer	2.93	-	55.88	-
3 layers	2.44	2.50	91.20	91.77
5 layers	2.5067	-	91.40	-

Figure 9: CNN Analysis

Accuracy in 3 CNN Layer model, with Kernel size= 5, has the best accuracy among all the Six CNN Models.

Model selection based on accuracy among ResNet-50 models (Based on Optimizers)

Hyperparameters selected are as follows:

- Cost Function: CrossEntropyLoss
- Learning Rate: 0.001
- Number of Epoch to find accuracy: 5

Model	Loss		Accuracy	
	ADAM	SGD	ADAM	SGD
	0.1259	**0.0581**	**96.249**	**98.226**

Figure 10: RESNET Results

Final Test Result:

Model	F1 Score	Accuracy
CNN 3 convolution layer	-	89.99
ResNet50	1	100

Figure 11: Final Test Results

The best-chosen model is: ResNet50 : Among Resnet and CNN, we see the best result in resnet, as it has 50 layers and more parameters that finetunes the output required.

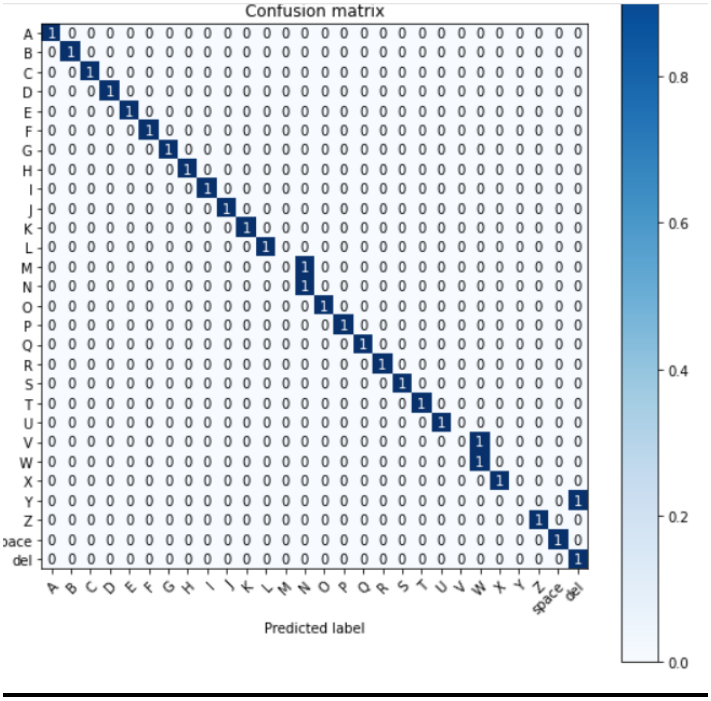

Figure 12:Confusion Matrix CNN

Hyperparameters are:

- Cost Function: CrossEntropyLoss
- Learning Rate: 0.001
- Number of Epoch to find accuracy: 5

Github Link: https://github.com/RudraxDave/AmericanSignLanguage_Reader

Future Scope:

The main challenge will be video-based sign Reader - to cope with the large variability of human hands. Capturing a scene and then taking certain frame from the data image captured- Optimize it and predict on it.

We can also work on more intense Models for the same, to make a Python based UI which uses Camera Application and Predicts on time the letter- which will in turn help everyone with Disability in Speaking or Listening convey their language. We can also work this in Low Light Conditions, or with our Dataset or any relatable different dataset.

Conclusion:

In this report, a functional Deep Learning based American sign language have been developed for asl alphabets. We achieved final accuracy of 99.0% with CNN on our dataset and also as the dataset for test is kind of little, we can get 100% Accuracy for RESNET50 model as well. We are able to improve our prediction after implementing different combinations of layers algorithms in which we verify and predict symbols which are more similar to each other.

Primary References and Codebase:

We built on the approach used in

1. Bantupalli, Kshitij & Xie, Ying. (2019). American Sign Language Recognition Using Machine Learning and Computer Vision.
2. C. N. Nyaga and R. D. Wario, "Sign Language Gesture Recognition through Computer Vision," 2018 IST-Africa Week Conference (IST-Africa), 2018, pp. Page 1 of 8-Page 8 of 8.
3. https://towardsdatascience.com/sign-language-recognition-using-deep-learning6549268c60bd
4. https://github.com/grassknoted/Unvoiced
5. http://karpathy.github.io/2019/04/25/recipe/
6. https://medium.com/@gryangalario/image-classification-on-the-american-sign-languagealphabet-dataset-45da66f8871e
7. https://public.roboflow.com/object-detection/american-sign-language-letters
8. https://github.com/dnmanveet/ASL-Alphabet-Prediction
9. Mohammed Waleed Kalous, Machine recognition of Auslan signs using PowerGloves: Towards large-lexicon recognition of sign language.
10. aeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convol utional-Neural-Networks-Part-2/
11. Pigou L., Dieleman S., Kindermans PJ., Schrauwen B. (2015) Sign Language Recognition Using Convolutional Neural Networks. In: Agapito L., Bronstein M., Rother C. (eds) Computer Vision - ECCV 2014 Workshops. ECCV 2014. Lecture Notes in Computer Science, vol 8925. Springer, Cham

12. Zaki, M.M., Shaheen, S.I.: Sign language recognition using a combination of new vision based features. Pattern Recognition Letters 32(4), 572–577 (2011)