# Computer Vision for American Sign Language

EE 541 - A Computational Introduction to Deep Learning

Nisha Elizabeth Jacob (USC ID : 8915179886)
(nejacob@usc.edu)
Muskaan Parmar (USC ID : 1660937440)
(muskaanp@usc.edu)

May 11, 2022

# Contents

# 1 Abstract

Despite the widespread use of sign language in recent years, it remains difficult for non-sign language speakers to communicate with sign language speakers. In recent years, advances in Computer Vision and Deep Learning have demonstrated promising progress in the fields of motion and gesture recognition. This would help enable communication with the hearing-impaired. We propose to create a model to read the American Sign Language using a dataset containing images from 29 classes. The training dataset contains 26 handshapes corresponding to the 26 letters of the alphabet and 3 other shapes to demonstrate Space, Delete and Nothing. We perform a comparison between multiple models with different architectures and model parameters to be able to determine the model that performs best on both, the provided dataset as well as on real world data.

# 2 Introduction

There has been an advance in recent studies to make it easier for impaired people to communicate since the number of hearing-disabled individuals has reached around 400 million according to the World Health Organization. Sign language is a visual communication system that allows people with hearing or speech impairments to communicate with each other and with others in their society. Fingerspelling is something that is often used in sign language. Fingerspelling is a method of spelling words using only hand gestures. One of the reasons the fingerspelling alphabet is so important in sign language is that signers use it to spell out names of things that don't have a sign. Hence, the recognition procedure for each individual letter is extremely important in its interpretation. There are multiple studies related to using motion sensors or the depth comparison of Microsoft Kinect to be able to interpret the feature vector and making use of algorithms such as k-Nearest Neighbors and Support Vector Machine for classification. With our proposed models, we aim to be able to accurately classify ASL hand sign images and compare the performance of previously unseen data on each model.

# 3 Dataset Description

The dataset has been obtained from Kaggle[1]. In total, there are 29 class labels: 26 alphabets(A to Z), space, delete and nothing. There are 3000 images of size 200x200 pixels per class label. The entire dataset has 87000 images which


Figure 1: Images from each class from the training set

have been split randomly as 80% for training and 20% for validation set. Thus, the training set has 69600 images and the validation set has 17400 images. The split between train and validation has been performed randomly. Figure 1 shows image from every class label from the training data. The test set provided by Kaggle has 28 test images only. We have also created our own dataset to check the model's performance on real world data. Our dataset has

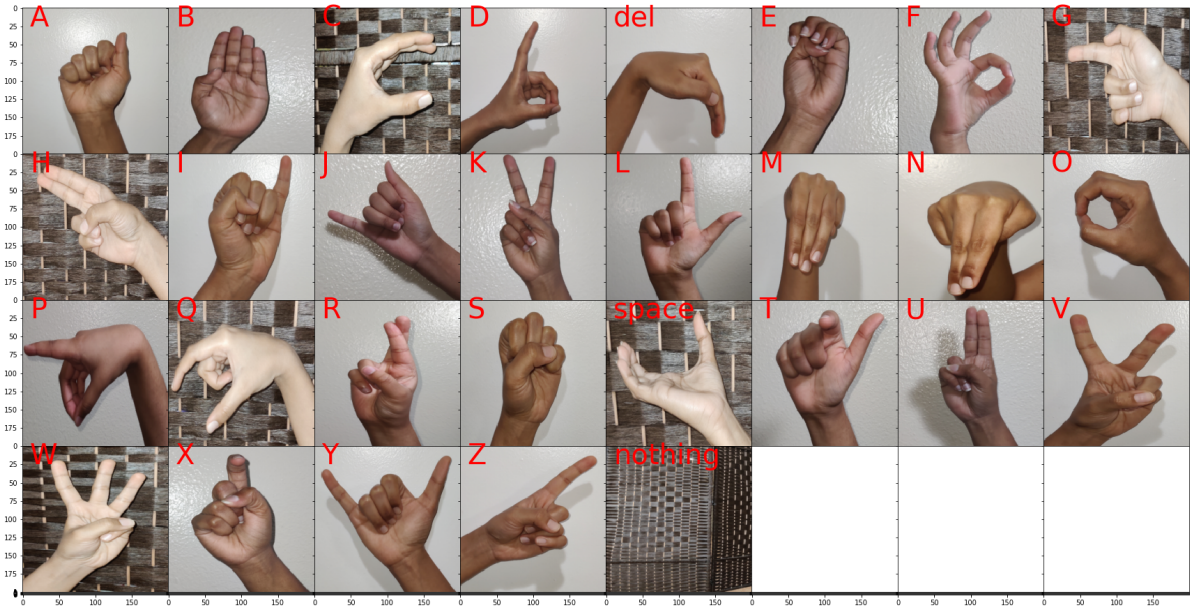



Figure 2: Images from each class from the testing set created

3 images per class label and has been taken against three different backgrounds to induce variety. In order to make our dataset comparable to the training data, we crop out the extra area around the handshapes. Thus, our dataset has 87 images. Figure 2 shows the images from our dataset.

We apply transformations like horizontal flip on the training data with a probability of 0.3. This helps the models to learn for the cases when the sign has been made from the left or right hand. Since neural networks work better on smaller images, rescaling them is an important step for building models. We rescaled the training, validation and test set images to 96x96. This size was found to be the appropriate one where there was no loss of features which would be required for classification. Finally, the images are converted to tensor format. The batch size is set to 128 for both training and validation.The selection of images for each batch is performed randomly.

# 4 Literature Review

In [2], the authors describe using a Convolutional Neural Network(CNN) for training on ASL images. They create a model from scratch by making use of Tensorflow and Keras libraries. Here, they also make use of the skin color of the hand in training the model and make use of the Convex Hull Algorithm to determine the shape as well as the location of the hand. The CNN model comprises input layer, two 2D convolution layers, pooling and flattening. ReLU activation is used for the hidden layers and Softmax for the output layer. They obtain an accuracy of approximately 98

For [3], the authors describe implementation of two different models for translating ASL images to words. They make use of a pretrained ResNet50 model and perform grid search for the best hyperparameters.The performance metric for the model was accuracy and Cross Entropy Loss. After obtaining the optimal hyperparameters, the model was trained finally on the entire dataset consisting of the train and the validation set.

In [4], the goal of the paper is to predict the hand gestures from every frame of a video. They make use of the MobileNet-V3 model and convert it into a 3D bottleneck by making use of three consecutive convolutional layers. They also use residual attention so as to increase the feature level semi-supervised learning. They make use of several dropout layers to avoid overfitting. The model is trained using weight decay regularization and SGD optimizer.The

learning rate is reduced by 0.01 after epoch number 25. Early stopping is also done to avoid overfitting after 30 epochs after validation loss keeps increasing continuously.

# 5 Different Models Implemented

The common set of training statistics used for all the models are described in Table 1.

| Parameter | Description |
|---|---|
| GPU | TESLA P100 |
| GPU Memory | 16GB |
| No. of CPU cores | 8 |
| Optimizer | Adam |
| Loss function | Multi Cross Entropy Loss |
| Batch Size | 128 |
| Time per Epoch | 10 minutes |
| Time to train | 4 hours |

Table 1: Training Statistics used for the models

## 5.1 Deep Neural Network

We have implemented a simple deep neural network as our first model. An example of an ANN is shown in Figure 3. Our model consists of 3 hidden layers with ReLU activations and a final output layer.

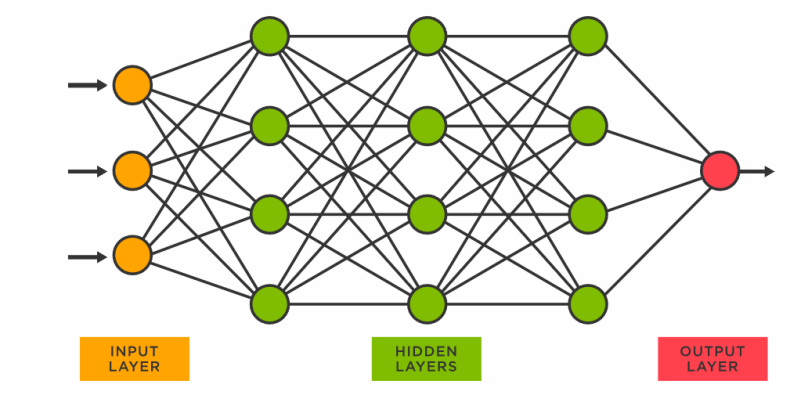



Figure 3: Layers of a Neural Network

The input and output dimensions for each layer are shown in Table 2.

| Layer Number | Input Dimension | Output Dimension |
|---|---|---|
| Layer 1 | 27,648 | 10,000 |
| Layer 2 | 10,000 | 2,000 |
| Layer 3 | 2,000 | 250 |
| Layer 4 | 250 | 29 |
| | **Trainable Parameters** | 296,999,529 |

Table 2: Input and Output dimensions for each layer in Neural Network

Kaiming Normal initialization was used to initialize the weights of each linear layer. The weights are initialized with values according to the method described in *Delving deep into rectifiers: Surpassing human-level performance*

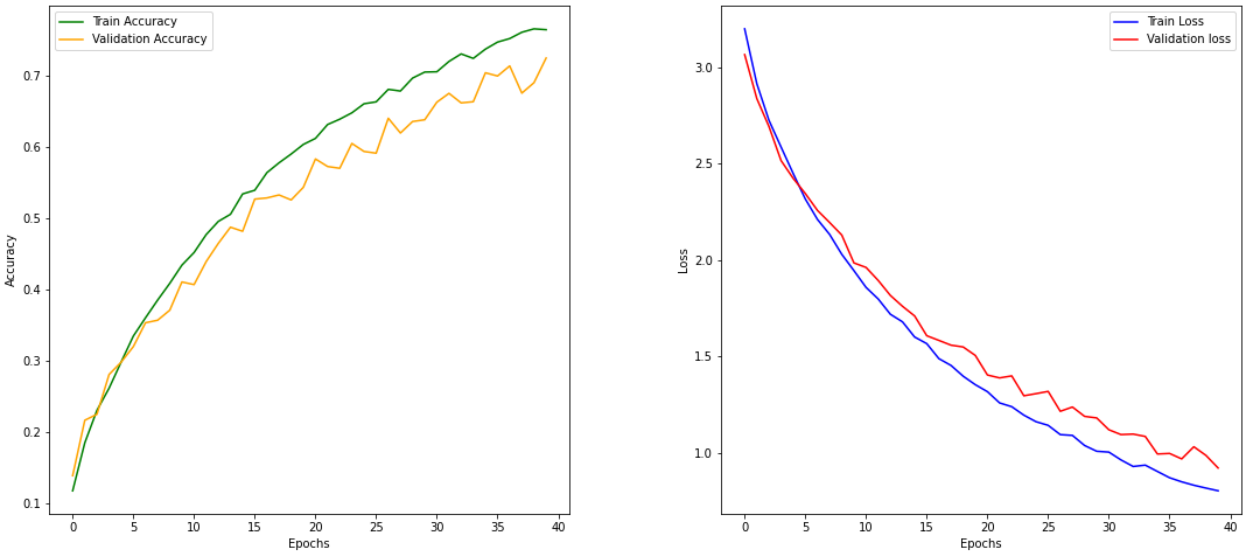
*on ImageNet classification* using a normal distribution. The non-linear function used with it is ReLU.

$$w_{i,j}^{(l)} = N(0; \frac{2}{N_{l-1}})$$

We utilize the multi category cross entropy criterion and use Adam as an optimizer with a regularization coefficient of $10^{-4}$. In order to be able to avoid overfitting, we also make use of early-stopping, where we stop training when validation starts performing consistently better than train data. We also experimented with different learning rates to be able to obtain the optimal value that gives best accuracy for the final model.

Initially, we try training on a small batch of data (7000 images) to be able to comprehend if the model architecture is able to learn to classify the alphabets from the training images before moving onto the entire train dataset. The training data is trained for around 40 epochs (or until early stopping occurs).

With a maximum learning rate of 0.00001 in Adam, we were able to achieve a final accuracy of 76.5% on the training data and an accuracy of 72.5% on the validation data. The loss on the training data was observed to be 0.8 and that on the validation data was 0.92. It can be inferred that the model does learn with this rate, but slowly. Accuracy on the provided test dataset is 82.14%. Accuracy on the self generated test data is 2.3%, meaning that our model is very poor at classifying real world data. Figure 4 highlights the accuracy and loss curves.



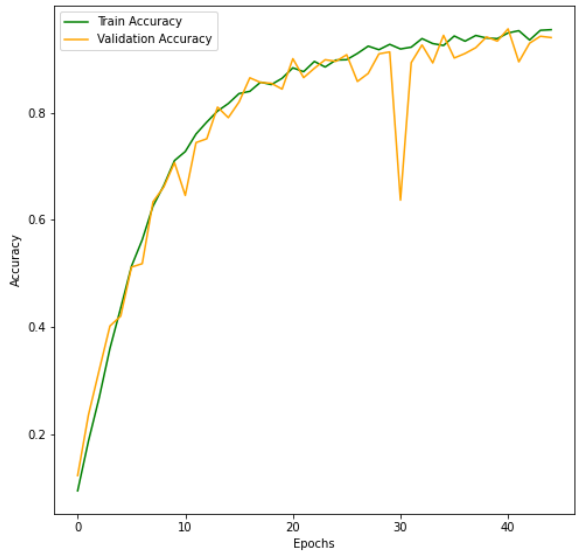

(a) Accuracy Curve

(b) Loss Curve

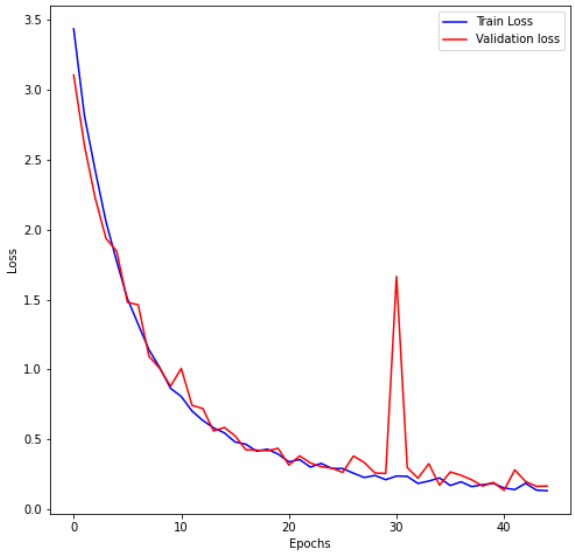
Figure 4: Accuracy and Loss Curves for both Training and Validation on Neural Network for learning rate of 0.00001

With a maximum learning rate of 0.0001 in Adam, we were able to achieve a final accuracy of 95.4% on the training data and an accuracy of 94% on the validation data. The loss on the training data was observed to be 0.13 and that on the validation data was 0.17. Accuracy on the provided test dataset is 89.28%. Hence, we can see that this model performs best on the given data. Accuracy on the self generated test data is 3.45%, meaning that even though our model can classify the original test data well, it is poor at classifying real world data. Figure 5 highlights the accuracy and loss curves.

With a maximum learning rate of 0.001 in Adam, we were able to achieve a final accuracy of 3.6% on the training data and an accuracy of 3.4% on the validation data. The loss on the training data was observed to be 3.37 and that on the validation data was also 3.37. It can be inferred that the model does not really learn with this learning rate since it is too high. Accuracy on the provided test dataset is 3.57%, meaning that only one image was correctly classified. Accuracy on the self generated test data is only 1.14%. Figure 6 highlights the accuracy and loss curves.

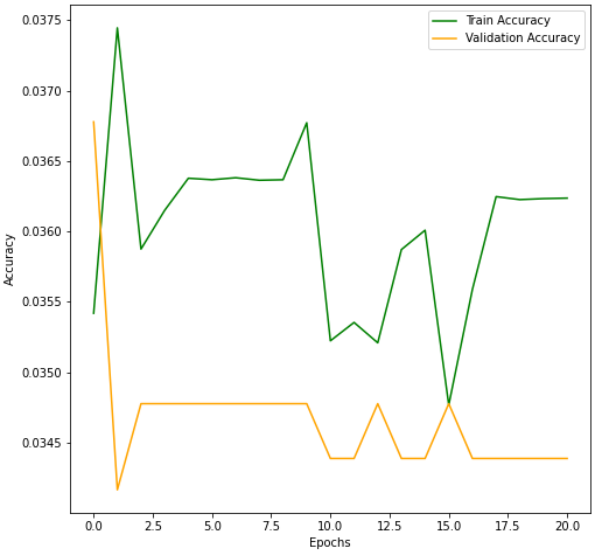



(a) Accuracy Curve

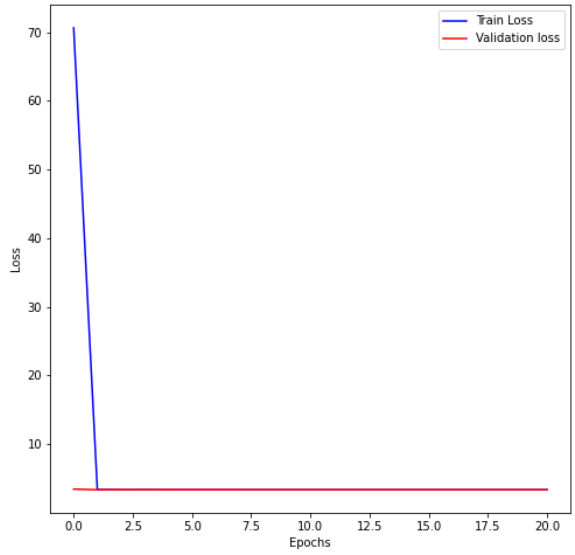



(b) Loss Curve

Figure 5: Accuracy and Loss Curves for both Training and Validation on Neural Network for learning rate of 0.0001




(a) Accuracy Curve




(b) Loss Curve

Figure 6: Accuracy and Loss Curves for both Training and Validation on Neural Network for learning rate of 0.001

## 5.2 Convolutional Neural Networks (CNN)

Convolutional Neural Networks are neural networks consisting of one or more convolutional layers. It is mainly used for image recognition, segmentation and classification. Kernels play an important role in the architecture of the CNN model. They are used to extract features from the input images by using convolution. The kernels can be thought of as a matrix which moves over the input image to extract useful information. There can be different types of kernels based on the type of extraction we wish to perform. A sample of the CNN model can be seen in Figure 7. We create a CNN model by making use of four convolutional layers followed by two linear layers and finally a

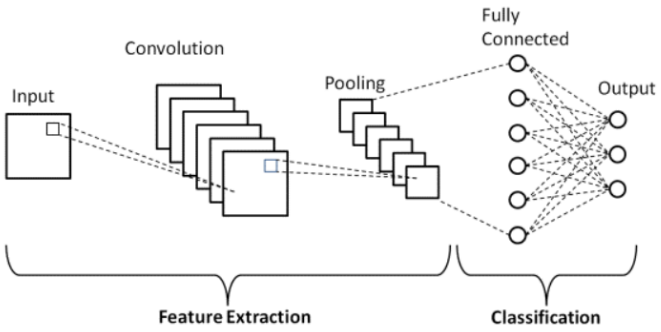



Figure 7: Sample of a CNN model

max pooling layer to predict the output label. In total, there are 71,789 training parameters in our model. The layers of the model have been described in Table 3. We use learning rate and weight decay of the Adam optimizer

| **Layer** | **Input Channel** | **Output Channel** | **Kernel Size** | **Stride** |
|---|---|---|---|---|
| Conv2d | 3 | 8 | (5,5) | (1,1) |
| Conv2d | 8 | 16 | (5,5) | (1,1) |
| Conv2d | 16 | 32 | (5,5) | (1,1) |
| Conv2d | 32 | 64 | (5,5) | (1,1) |
| Linear | 64 | 128 | - | - |
| Linear | 128 | 29 | - | - |
| MaxPool2d | 29 | 29 | (2,2) | (2,2) |
| **Total Trainable Parameters** | **71,789** | | | |

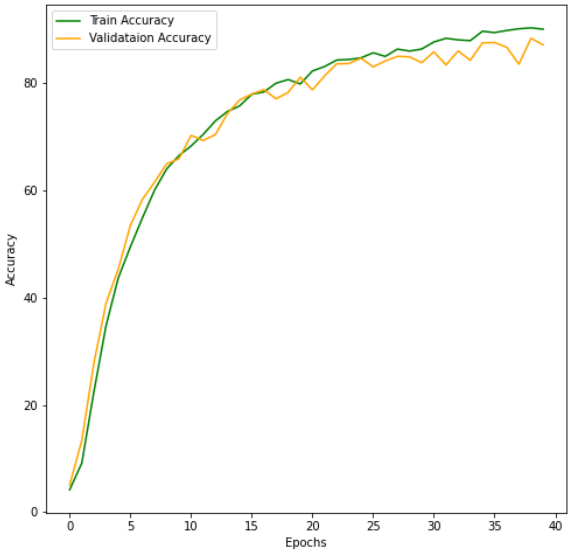
Table 3: Training layers of the CNN network

as hyperparameters. Weight decays help to prevent overfitting and is used during weight update step while training. We make use of Cross Entropy Loss as our criterion function. Cross Entropy Loss is preferred as it works well for multiclass classification problems and compares the predicted class probability with the true class, and averages the loss across all the images present in testing dataset. The optimal values of learning rate and weight decay should give the best validation accuracy as compared to other cases. We try out the following values of learning rate: 0.001 and 0.01 and the following values were considered for weight decay: 0.0001 and 0.001. We also found the accuracy of each model on the two test sets. A good accuracy on the test set would show that the model is able to correctly classify images from an unknown set as well.

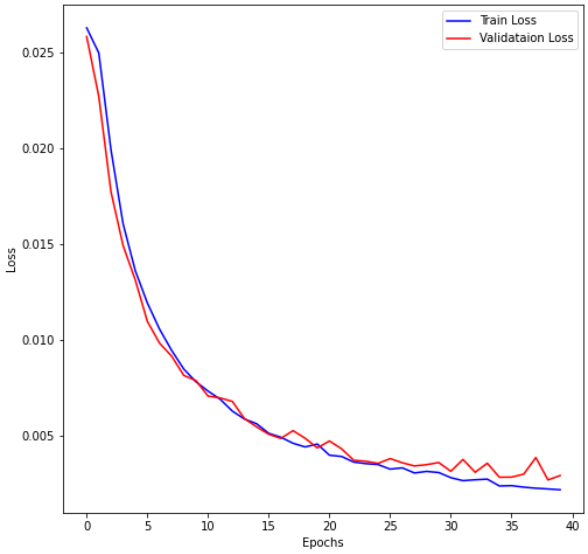
Figure 8 and Figure 9 show that for learning rate of $10^{-2}$ and for weight decay of $10^{-3}$ and $10^{-4}$, the model is able to train properly and thus performs really well on the validation set. This is also confirmed by the loss curves seen for both the values of weight decay where the loss for validation is continuously decreasing.

Figure 10 and 11 show that for learning rate of $10^{-3}$ and for weight decay of $10^{-3}$ and $10^{-4}$, the model is unable to train properly and thus performs poorly on the validation set. This is also confirmed by the loss curves seen for both the values of weight decay where the loss has spikes instead of decreasing gradually. It shows that even after training the model for 40 epochs the model is unable to extract the features.

Table 4 shows the variations in accuracy and loss for both the training and validation datasets for different combinations of the two hyperparameters.

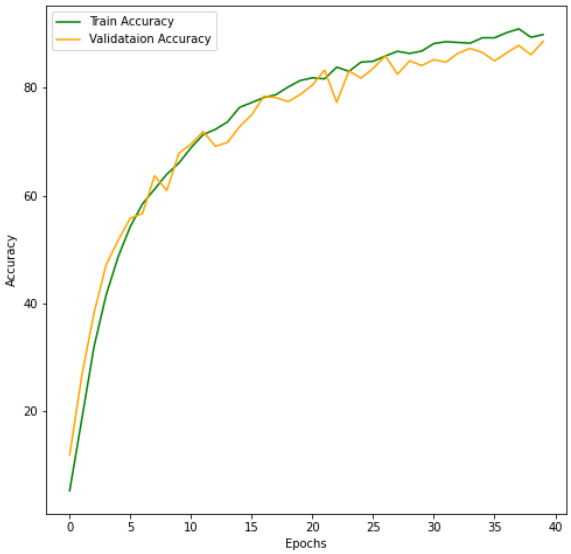



(a) Accuracy Curve

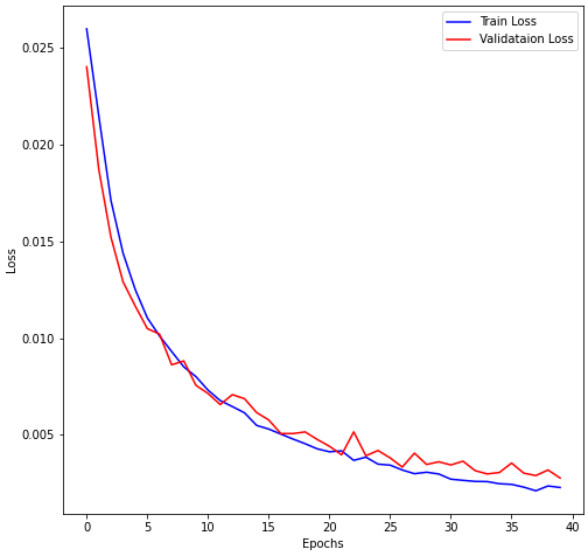



(b) Loss Curve

Figure 8: Accuracy and Loss Curves for both Training and Validation on CNN for learning rate of $10^{-3}$ and weight decay of $10^{-3}$

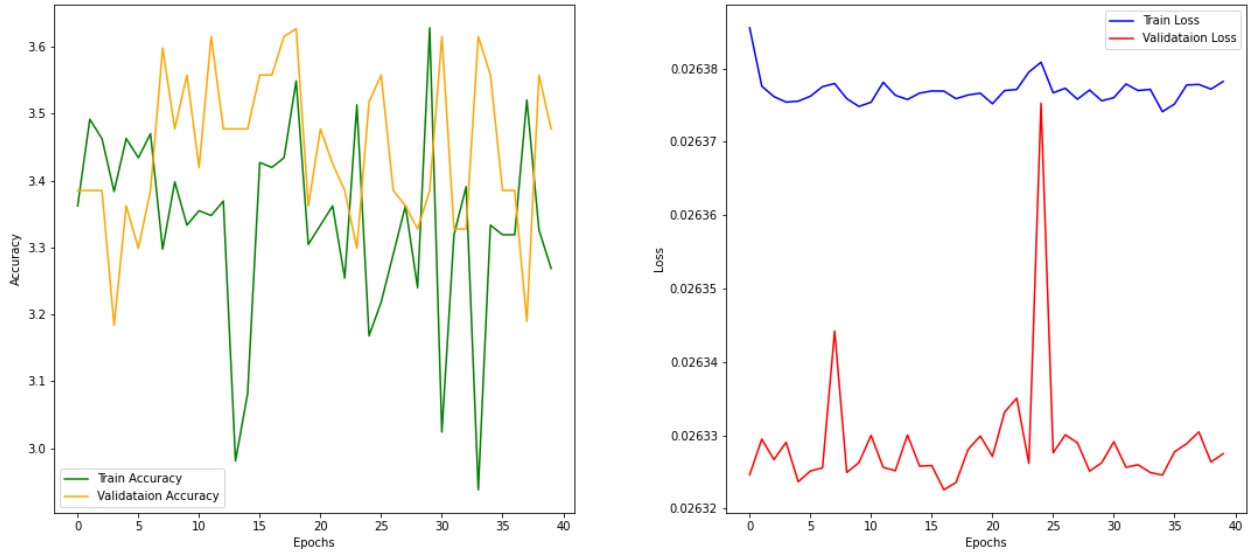



(a) Accuracy Curve




(b) Loss Curve

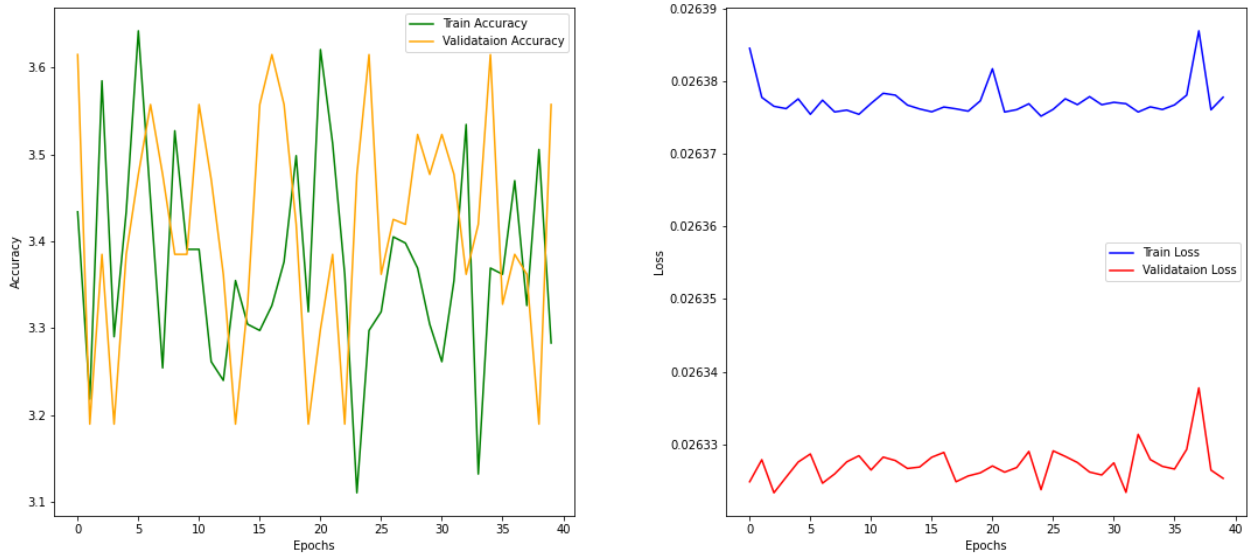
Figure 9: Accuracy and Loss Curves for both Training and Validation on CNN for learning rate of $10^{-3}$ and weight decay of $10^{-4}$




(a) Accuracy Curve

(b) Loss Curve

Figure 10: Accuracy and Loss Curves for both Training and Validation on CNN for learning rate of $10^{-2}$ and weight decay of $10^{-4}$




(a) Accuracy Curve

(b) Loss Curve

Figure 11: Accuracy and Loss Curves for both Training and Validation on CNN for learning rate of $10^{-2}$ and weight decay of $10^{-3}$

| **Performance** | **LR**=$10^{-2}$, **WD**=$10^{-3}$ | **LR**=$10^{-2}$, **WD**=$10^{-4}$ | **LR**=$10^{-3}$, **WD**=$10^{-3}$ | **LR**=$10^{-3}$, **WD**=$10^{-4}$ |
|---|---|---|---|---|
| Train Accuracy | 90.04% | 89.83% | 3.27% | 3.28% |
| Validation Accuracy | 87.11% | 88.56% | 3.48% | 3.56% |
| Train Loss | 0.0022 | 0.0023 | 0.0264 | 0.0264 |
| Validation Loss | 0.0029 | 0.0028 | 0.0263 | 0.0263 |
| Test Accuracy (Kaggle) | 92.85% | 78.57% | 3.57% | 3.57% |
| Test Accuracy (self generated) | 27.38% | 26% | 3.57% | 3.57% |

Table 4: Comparing the performance of CNN using different learning rates and weight decay

From the Table 4 values we can see that the CNN model with learning rate $10^{-2}$ and weight decay of $10^{-3}$ has a good accuracy on the Kaggle dataset as compared to the other models.The model with learning rate $10^{-2}$ and weight decay of $10^{-4}$ has its train accuracy close to the best performing model but the performance on test set(Kaggle) is lower than the best performing model.

### 5.3 ResNet-50

ResNet stands for Residual Network. It was developed in order to alleviate the problem of training deeper networks by introducing a new neural network layer - the Residual Block. It uses skip connections between layers (as shown in Figure 12) that add the outputs of previous layers to the outputs of future stacked layers. This helps in reducing the problem of vanishing gradient and allows the learning of alternative networks.

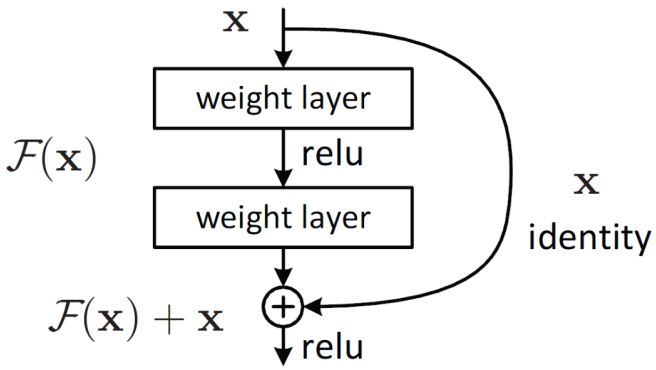



Figure 12: Skip Connection in ResNet

ResNet-50 is a convolutional neural network that is 50 layers deep. It was inspired by VGG-16/19 neural networks with the convolutional networks having 3x3 filters. Resnets have comparatively lower complexity and fewer filters than VGGNets. The layers here have the same number of filters for the same output feature map size. They are doubled if the feature map size is halved in size to preserve time complexity per layer. PyTorch provides ResNet-50 as a pre-trained model, that has been trained on more than a million images from the ImageNet database. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil and many animals. We

| Layer Number | Input Dimension | Output Dimension |
|---|---|---|
| Layer 1 | 2,048 | 1,000 |
| Layer 2 | 1,000 | 500 |
| Layer 3 | 500 | 29 |
| Layer 4 | 250 | 29 |
| | **Total Trainable Parameters** | 2,564,029 |

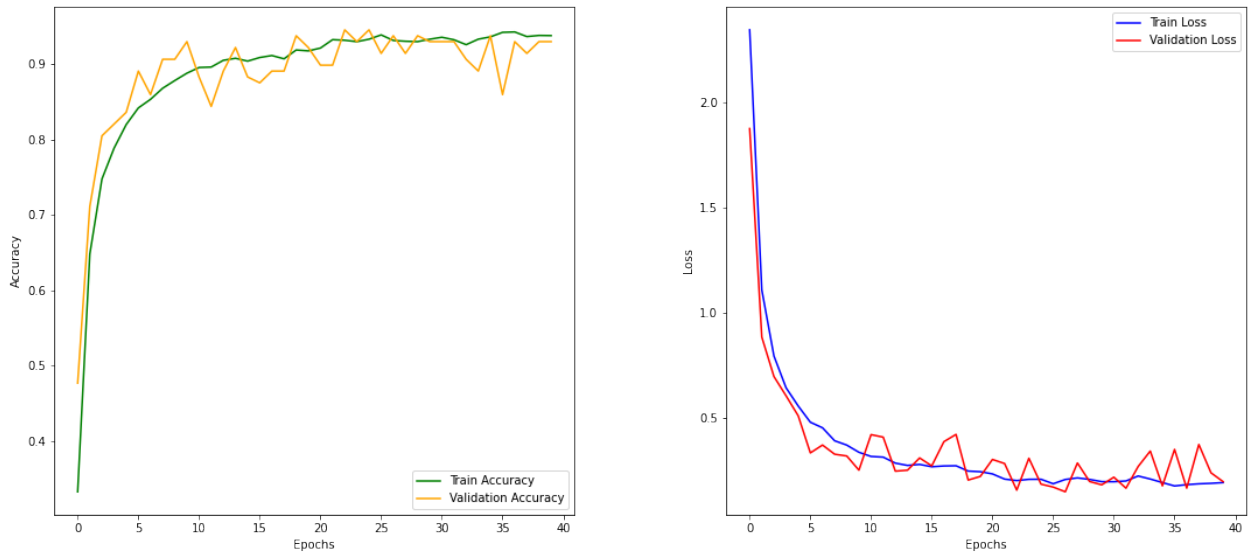
Table 5: Input and Output dimensions for Fully Connected layer in Resnet-50

utilize transfer learning to finetune the available model to be able to classify images from our dataset. The feature

extraction layers of the network are frozen, so that they are not back-propagated through, during training. The fully connected layer of the network is re-defined and trained using our dataset. We add two hidden layers with ReLU activations and a dropout probability of 0.3 and 0.2 respectively for each layer. The input and output dimensions for each layer are as shown in the Table 5.

We see that there is a huge reduction in the total number of trainable parameters in Resnet as compared to CNN. This is because each filter is used many times over the input feature map (parameter reuse) and sparse connectivity between layers. Multi category cross entropy loss and Adam (with a regularization coefficient of $10^-4$) are used as the criterion and optimizer for the fully connected layer respectively. Early stopping is used here as well, in order to avoid overfitting. We first try training on a small batch of data to be able to comprehend if the model architecture is able to learn to classify the alphabets from the training images before moving onto the entire train dataset. The training data of size 69000 is trained for around 35 epochs (or until early stopping occurs). Since this is a pre-trained model where we do not need to backprop through the feature extraction layers, execution is significantly faster with each epoch taking around 5 minutes, making the total time to train, 2 hours.

With a maximum learning rate of 0.001 in Adam, we were able to achieve a final accuracy of 93.8% on the training data and an accuracy of 93% on the validation data. The loss on the training data was observed to be 0.19 and that on the validation data was 0.3. Accuracy on the provided test dataset is 92.85%. This model performs well on the given data. Accuracy on the self generated test data is 22.9%. This is comparatively better than the accuracy obtained with Neural Network. Figure 13 highlights the accuracy and loss curves.



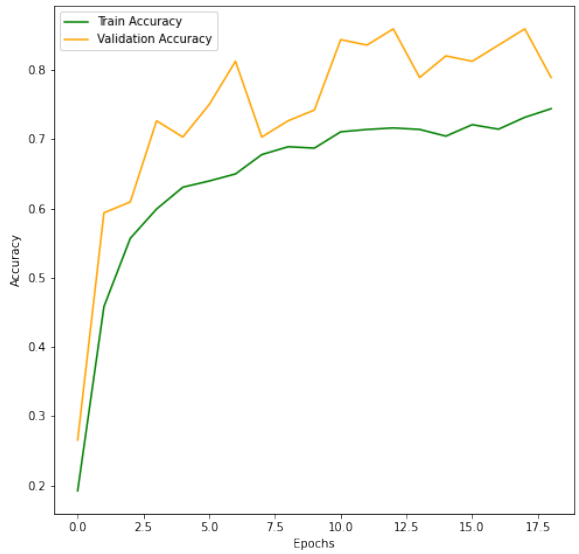

(a) Accuracy Curve

(b) Loss Curve

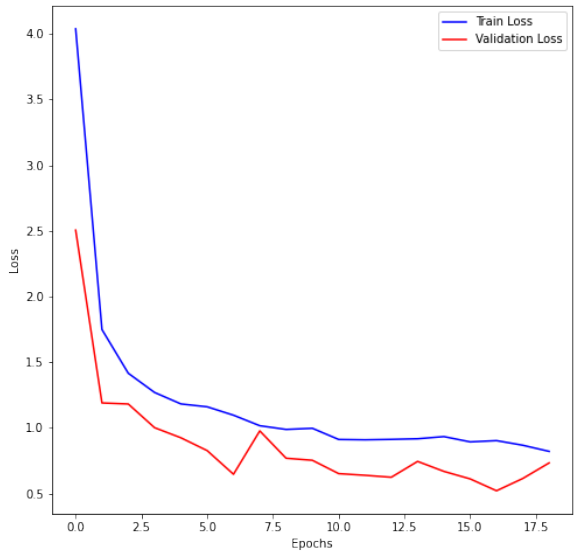
Figure 13: Accuracy and Loss Curves for both Training and Validation on ResNet-50 for learning rate of 0.001

With a maximum learning rate of 0.01 in Adam, we were able to achieve a final accuracy of 74.4% on the training data and an accuracy of 78.9% on the validation data. The loss on the training data was observed to be 0.82 and that on the validation data was 0.73. It can be inferred that the model does learn with this rate, but slowly. Accuracy on the provided test dataset is 85.71%. Accuracy on the self generated test data is 13%. Our model is still able to classify images, but with a poorer accuracy. Figure 14 highlights the accuracy and loss curves.

With a maximum learning rate of 0.1 in Adam, we were able to achieve a final accuracy of 3.3% on the training data and an accuracy of 2.3% on the validation data. The loss on the training data was observed to be 3.4 and that on the validation data was 3.37. It can be inferred that the model does not really learn with this learning rate since it is too high. Accuracy on the provided test dataset is 3.57%, meaning that only one image was correctly classified.

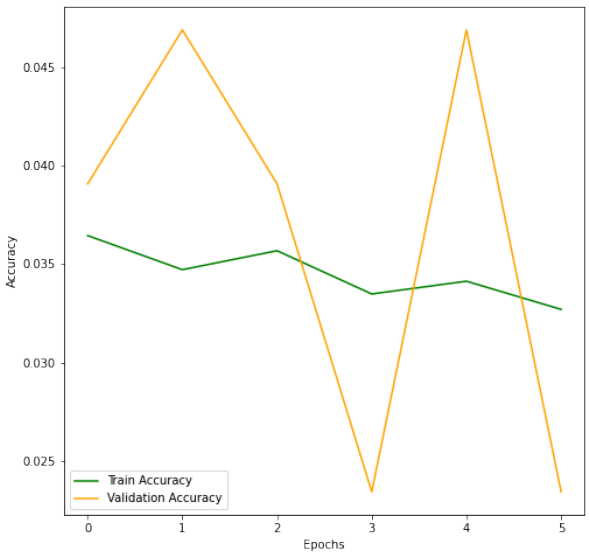



(a) Accuracy Curve

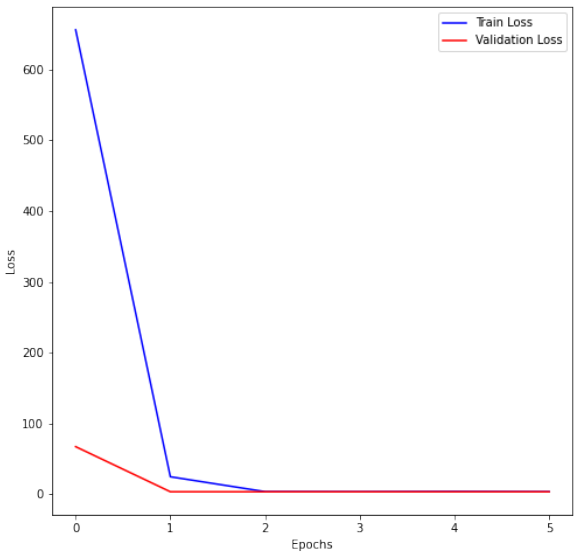



(b) Loss Curve

Figure 14: Accuracy and Loss Curves for both Training and Validation on ResNet-50 for learning rate of 0.01




(a) Accuracy Curve




(b) Loss Curve

Figure 15: Accuracy and Loss Curves for both Training and Validation on ResNet-50 for learning rate of 0.1

Accuracy on the self generated test data is only 1.14%. This model does not work well on both test datasets. Figure 15 highlights the accuracy and loss curves.

Table 6 depicts the performance of the ResNet model on different learning rates and the test set.

| **Performance** | **LR**=$10^{-3}$ | **LR**=$10^{-2}$ | **LR**=$10^{-1}$ |
|---|---|---|---|
| Train Accuracy | 93.8% | 74.4% | 3.3% |
| Val Accuracy | 93% | 78.9% | 2.3% |
| Train Loss | 0.19 | 0.82 | 3.4 |
| Val Loss | 0.3 | 0.73 | 3.37 |
| Test Accuracy(Kaggle) | 92.85% | 85.71% | 3.57% |
| Test Accuracy(self generated) | 22.9% | 13% | 1.14% |

Table 6: Comparing the performance of ResNet model using different learning rates

## 5.4 MobileNet-V2 Model

Mobile Net is a type of CNN model open-sourced by Google and works great on problems focusing on image classification. It was created for mobile applications. Its popularity is due to the fact that the network has less trainable parameters as compared to other neural network models with the same depth. This gives rise to a lightweight and deep neural network.

PyTorch provides us with pretrained models. We make use of the MobileNetV2 model provided in PyTorch and retrain the fully connected part of it. The MobileNet-V2 model has 50 layers. We made changes to the classifier layer of the model by adding two extra hidden layers with ReLU activations and dropouts of 0.2 probability. The feature extraction layers are frozen so that no back propagation is performed on those layers.

We make use of Cross Entropy Loss as our criterion function. Also, we take the learning rate of Adam optimizer as a hyper parameter. We try out the following values of learning rate: 0.001, 0.01 and 0.1. The best performing model is selected on the basis of the performance on the validation set. The model is trained for 40 epochs and early stopping is performed when the validation loss for the current epoch exceeds the validation loss for the previous epoch consecutively for three times. From the plots in Figure 16, we can see that the model is able to learn on the training dataset when the learning rate is set to 0.001. We can see that the training accuracy comes up around 99%. Also, the loss curve is decreasing gradually for both the validation and the train dataset.

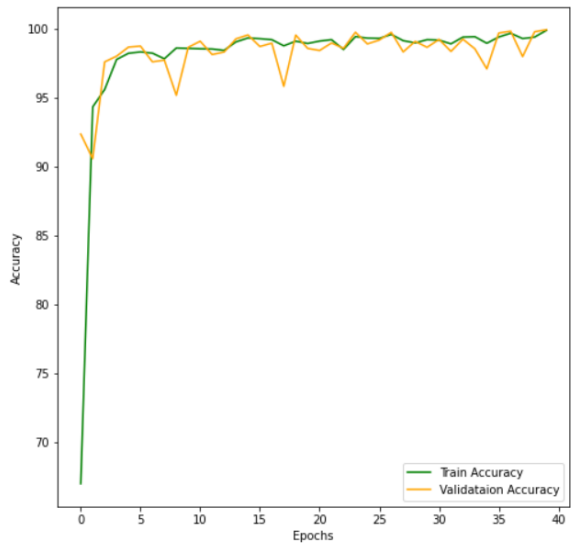
When the learning rate is set to 0.01, we can see in Figure 17 that the accuracy on both the training and validation set reaches around 40% after the completion of 40 epochs.

For a learning rate of 0.1, the model performs poorly as it is unable to train on the training data. We can say this by looking at the Figure 18, where the accuracy curves for both train and validation are increasing and decreasing suddenly instead of a gradual increase. This is also confirmed by the loss curves where there is no reduction visible.

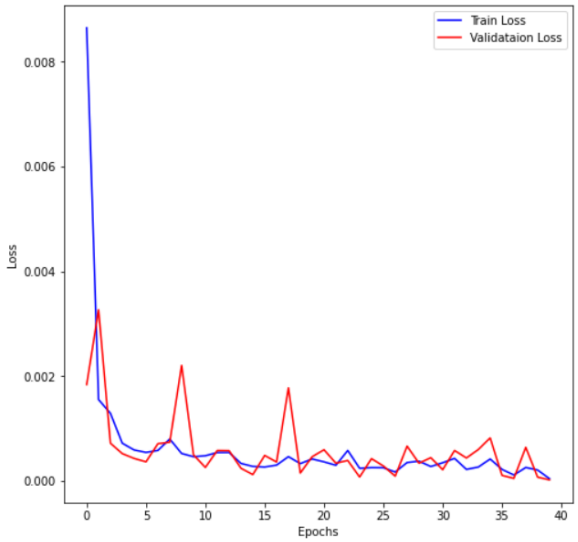
Table 7 below shows the performance of the MobileNet-V2 model on different learning rates as well as on the test set. From the table, we can say that the model with the learning rate of 0.001 performs the best.

| **Performance** | **LR**=$10^{-3}$ | **LR**=$10^{-2}$ | **LR**=$10^{-1}$ |
|---|---|---|---|
| Train Accuracy | 99.22% | 36.62% | 3.25% |
| Val Accuracy | 99.78% | 44.84% | 3.47% |
| Train Loss | 0.0003 | 0.0134 | 0.0264 |
| Val Loss | 0.0001 | 0.0117 | 0.028 |
| Test Accuracy(Kaggle) | 96.4 | 53.57 | 3.57 |
| Test Accuracy(self generated) | 51.19 | 3.57 | 3.46 |

Table 7: Comparing the performance of MobileNet-V2 model using different learning rates

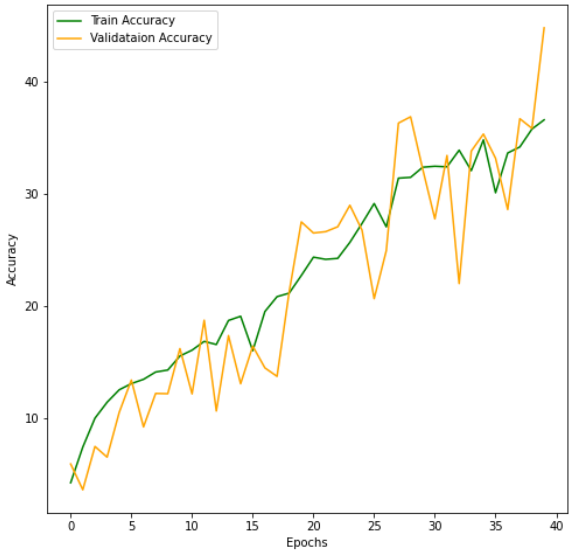



(a) Accuracy Curve

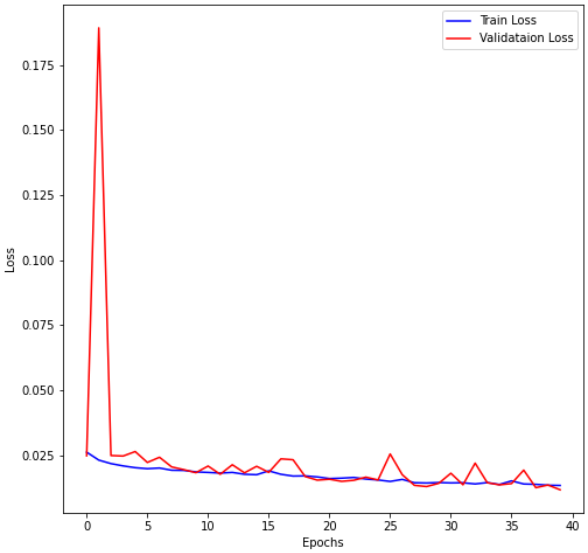



(b) Loss Curve

Figure 16: Accuracy and Loss Curves for both Training and Validation on MobileNet-V2 for learning rate of 0.001

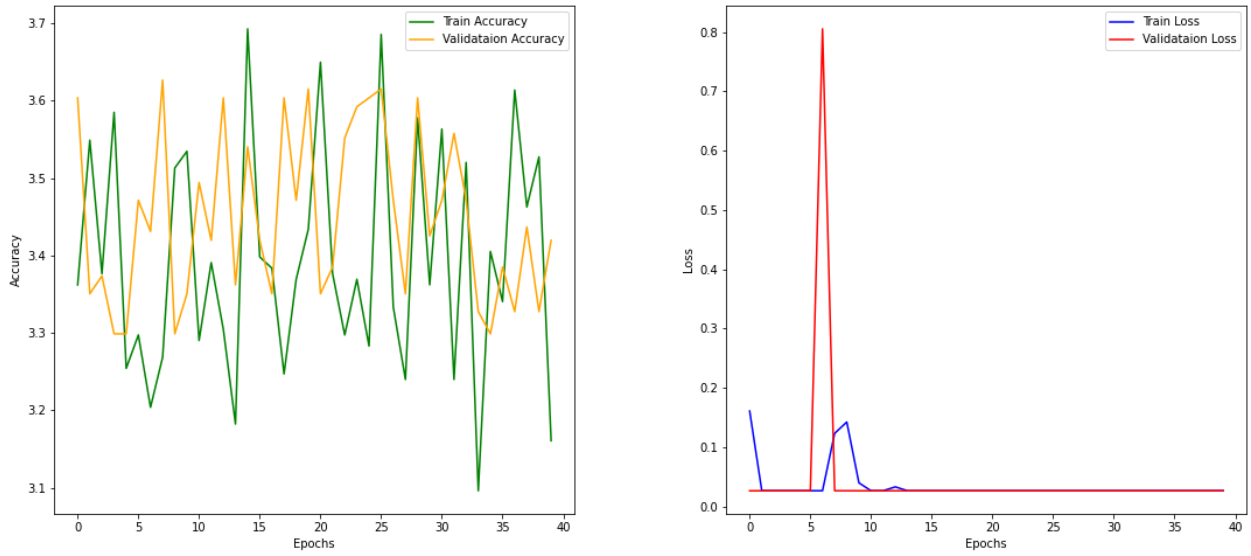



(a) Accuracy Curve




(b) Loss Curve

Figure 17: Accuracy and Loss Curves for both Training and Validation on MobileNet-V2 for learning rate of 0.01




(a) Accuracy Curve

(b) Loss Curve

Figure 18: Accuracy and Loss Curves for both Training and Validation on MobileNet-V2 for learning rate of 0.1

# 6 Comparison of Results

Table 8 specifies the accuracy of each optimal model on the train, validation and test (provided and self generated) data.

We see that MobileNet-V2 performs best on both the test datasets. It misclassifies one image from the provided test data and is able to manage an impressive accuracy of 51% on the real world test data. None of the other models are able to achieve more than 30% on the self generated data. This could be because there was no similar training data provided to the models due to which they are unable to accurately decipher the hand gestures. Our CNN model

| **Network** | **Optimal Learning Rate** | **Train Data** | **Validation Data** | **Test Data** | **Self Test Data** |
|---|---|---|---|---|---|
| Neural Network | $10^{-4}$ | 95.4% | 94% | 89.28% | 3.45% |
| Resnet-50 | $10^{-3}$ | 93.8% | 93% | 92.85% | 22.9% |
| MobileNet-V2 | $10^{-3}$ | 99.2% | 99.78% | **96.4%** | **51.19%** |
| CNN | $10^{-2}$, weight decay = $10^{-3}$ | 90.04% | 87.11% | 92.85% | 27.38% |

Table 8: Comparing the performance of each optimal network on the given data

is also able to provide a reasonably good accuracy on the self generated test data as compared to the other models. The Resnet pretrained model is unsuccessful in interpreting the same data, but it compares with CNN in predicting the original test data.

We see that our Neural Network is able to comprehend the original test data very well (almost similar to CNN accuracy), but it performs poorly on the real world data, since it can only decipher 3 of the given 87 images.

Since neural networks always find the easiest way to solve a problem, we need to include maximum diversity in our dataset and maximize the coverage (by training on extremely difficult examples) to be able to achieve appreciable accuracies on real world data.

# 7 Challenges

- *Generating Test Data:* Since the test dataset for the problem is very small, we had to generate our own test data to determine each model's performance and accuracy on real world data. The images have to be taken with varying backgrounds to enable diversity. Other pre-processing techniques such as cropping are also required to be able to eliminate unwanted details in the image. Since there are many output classes, it is time consuming to manually capture multiple images for each class.
- *Size of the Training Dataset:* Since the training data consists of 69,600 200x200 RGB images, it takes a significant amount of time for the model to learn from each image for around 40 epochs. Using a CPU alone would cause the execution times to be much longer, hence, GPUs are required for the task.

# 8 Future Work

- Generate better and larger variety of images for both training and test data
- Build further on the model by providing live transcription of captured video feed. This transcription can then be used to produce speech.

# 9 Conclusion

For this project, we tried comparing the performances of four deep learning models - simple Neural Network, Convolutional Neural Network, pretrained ResNet-50 network and a pretrained MobileNet V2 network. Hyperparameter tuning was performed for each model to obtain the best architecture. It is observed that the MobileNet network performs best on the given ASL dataset with an accuracy of 51.19% on real world test data.

# References


[1] *ASL Alphabet Dataset*, available at `https://www.kaggle.com/datasets/grassknoted/asl-alphabet`

[2] Murat Taskiran, Mehmet Killioglu, Nihan Kahraman, *A Real-Time System for Recognition of American Sign Language by using Deep Learning*

[3] Nihar Joshi, Ryan Gudal, Tianyu Jiang, Samantha Clark, *American Sign Language Recognition Using Computer Vision*

[4] Evgeny Izutov, *ASL Recognition with Metric-Learning based Lightweight Network*

[5] Jason Atwood, Matthew Eicholtz, Justin Farrell, *American Sign Language Recognition System*

[6] Sujay S Kumar, Tenzin Wangyal, Varun Saboo and Ramamoorthy Srinath, *Time Series Neural Networks for Real Time Sign Language Translation*

[7] Vivek Bheda, N. Dianna Radpour, *Using Deep Convolutional Networks for Gesture Recognition in American Sign Language*

[8] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen, *MobileNetV2: Inverted Residuals and Linear Bottlenecks*

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, *Deep Residual Learning for Image Recognition*

[10] Andrej Karpathy, *A Recipe for Training Neural Networks*, available at `http://karpathy.github.io/2019/04/25/recipe/`

[11] *Image classification (American Sign language) using PyTorch*, available at `https://blog.jovian.ai/image-classification-american-sign-language-using-pytorch-7bef9d7e8a25?gi=c2dd6cb441d1`