EE 541 – Computational Introduction to Deep Learning

CV for American Sign Language

Xinlei Yu, Xingjue Liao

May 8, 2023

Abstract

In this paper, we implement an alphabet translator for American Sign Language (ASL). To assist with the task, we use both Convolutional Neural Networks (CNN) and Residual Neural Network (ResNet) to classify RGB images of ASL alphabet hand gestures, after we trained our models using a large pre-processed dataset. For both models, we have tuned many hyper-parameters such as number of layers in CNN and optimizer learning rates to achieve the best results. In the end, our models have very high training accuracy and acceptable test accuracy.

Contents

1 Introduction

About 15% of the US population suffer from hearing losses of some degrees[6], and for those who are deaf or hard of hearing, they often use American Sign Language (ASL) which is expressed by movements of the hands and face to communicate with others. Today, ASL is the natural language of around 500,000[12] people. To help them to better communicate with those who are not proficient with ASL, we found there's a need to build a translator that can generate words as captions from the finger spelling of ASL users.

Recent developments of machine learning have impacts throughout all aspects of our lives, and computer vision applications such as RGB image classification have become increasingly popular. We have chosen this project in the hope that our ML model may assist people with hearing loss by providing an automated framework to transcribe American Sign Language (ASL) into written language. Our first step is building a model that can recognize alphabet letters from ASL hand signals.

2 Problem Statement

Deep Learning for American Sign Language (ASL) recognition has gained significant attention in recent years, with researchers using various data sets available on Kaggle to develop innovative models. Our goal is the implemented deep learning model able to classify the test image into correct class. The detail of the training sets and test sets will be discussed in Data Set section.

We plan to first implement our own version of CNN (starting with 2 layers). Then we'll replace our CNN with a more complicated ResNet model to see whether it has better performances. We noticed the Kaggle test set is small and we planned to create our own test set. When we began our project, we decided if any of our model's test accuracy is above 80 percent on our own test set, we'll call the project a success.

3 Previous Research

Numerous authors have tried to implement this task: in[8], the authors built their own CNN, and employed Caffe, a deep learning framework, to train the data from the ASL FingerSpelling Dataset. They are able to produce a robust model for letters A-E, and a modest one for letters A-K for image input. Another work was done by [10], in which the authors used a modified Squeezenet[9] model, and expand the alphabet recognition to all letters. For the Kaggle dataset we plan to use, there's an online article [4] in which the author built his own CNN and achieved satisfying results. He mentioned the result can be improved if the model is replaced by Mask R-CNN or RetinaNet. Finally, this online article[1] built an ASL alphabet recognition system using pre-trained ResNet-50 from the fast.ai library, and implement real-time prediction using OpenCV that it can read real-time webcam video input.

One of most significant technical difficulty noticed by these previous works is that the trained model is easily over-fitted, as the dataset pictures usually have highly similar backgrounds (especially those in the Kaggle dataset we plan to use). In addition, ASL is a natural language and its alphabet is not 'scientifically designed', which cause many letters to be highly similar, such as 'M','N','S','T', shown in the picture below. Such subtle difference between letters requires sophisticated feature extraction layers, and therefore making the model prone to over-fitting. We realized that we might need to make a balance between training accuracy and robustness against over-fitting.

In addition, we found the task also related to hand gesture recognition in general, in [11] and [14], we learned how a similar task can be done using CNN. Compared to classical image classification approaches, these hand gesture recognition papers used simpler models and put more efforts into

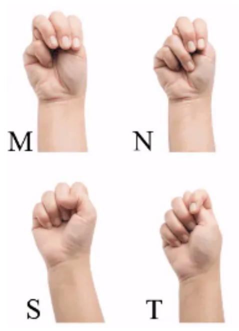


Figure 1: Similar letters requires sophisticated feature extraction.[5]

image de-noising techniques and try to downplay the effects of the variation of the light and image background.

In the end, we found that some letters (ex. 'Z') in ASL alphabet are spelled by continuous motions, so we were interested to achieve temporal motion recognition by using video input, as shown in [13]. However, it went beyond the scope of the project, but it provides interesting topics for future exploration.

4 Data Set

ASL is a natural language that serves as a predominant sign language for the Deaf people in the United States of America.

4.1 DataSet from Kaggle

The data set is made available by Akash Nagaraj from Kaggle. The link is linked here for more information.

- The dataset linked above contains images from 29 classes (26 alphabets, SPACE, DELETE and NOTHING).
- Each training set contains 3000 images in the training set and each image is a 200 x 200 RGB image.
- Each test set contains one image that does not seen by the trained model before. If the trained model classify every test images into correct class, we could conclude the Deep Learning model is accurate.

Below are some of examples images used to deep learning model.

4.2 Additional Data Set

Besides the mere test sets provided from the Kaggle, we added additional test sets to test the deep learning models. An example is showing below. For the new test sets, we found out it has poor accuracy. We realize the new test sets has very different background as the training sets. This might lead the poor accuracy. To improve the accuracy, we introduce the high contrast test data sets.

5 Implementations

In this project report, we will spend two sections to present our project methods and discussion.

Figure 2: Example Image of Train Set of Class Nothing.

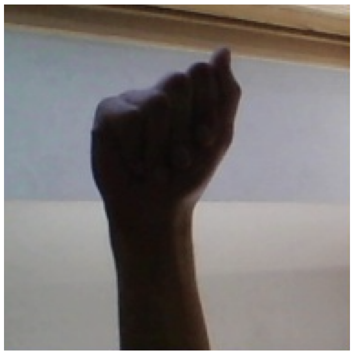


Figure 3: Example Image of Train Set of Class A.

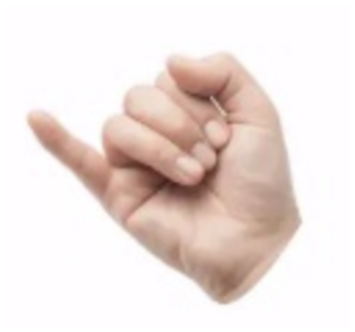


Figure 4: Example Image of New Test Set of Class J.[5]

5.1 CNN

A Convolution Neural Network (CNN) is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other. In general, CNNs are great at finding patterns and then using those to classify images. Typically, a CNN has 3 types of layers:

- Convolutional Layer: the layer consists of learnable filters that activate when they see some types of visual features such as a blotch of some color or an edge.
- Pooling Layer: pooling layers are used to decrease the size of the convoluted feature maps and reduce the computational costs.

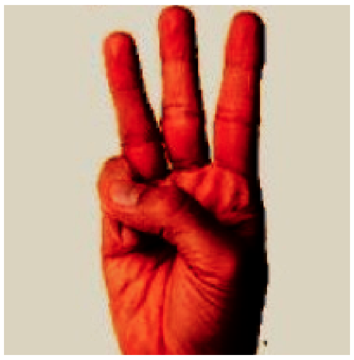


Figure 5: Example Image of Contrast Test Set of Class W.[2]

- Fully Connected Layer: these layers utilize the output from the previous convolution process and predict the class of the image based on the features extracted in previous stage. A typical CNN architecture is shown in figure below:

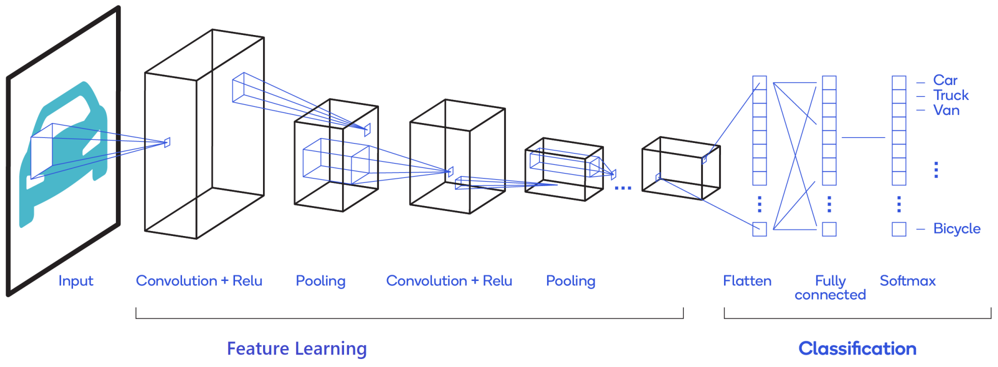


Figure 6: A typical CNN architecture [3]

The following is the hyperparameters we tried during the trials.

- Model Architecture: CNN
- Number of Conv layers: [2, 4]
- Size of the kernel: [3*3, 5*5]
- Loss Function: Cross Entropy Loss.
- Regularization: L2 with weight decay 0.0001
- Optimizer: Adam
- Learning Rate: [0.0005, 0.001]
- Number of Epochs: [5, 10]

We settled down with CNN architecture with 4 convolutional layers, shown in the figure 7 below. By increasing the number of conv layers from 2 to 4, we can greatly increase the training and testing accuracy as the 4-layer-CNN can capture more sophisticated features.

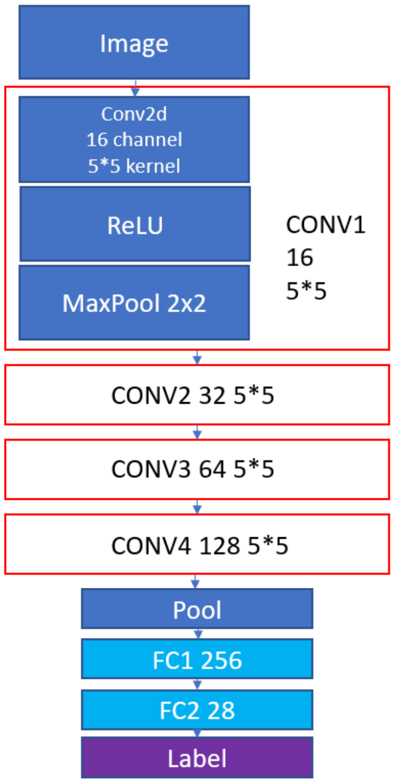


Figure 7: The custom CNN architecture

5.2 ResNet

In the project, we also design an implementation using Residual Neural Network(ResNet), which is a deep learning model typically used for computer vision application.ResNet was specifically designed to address the problem of vanishing gradients in deep neural networks. This architecture has significantly enhanced the performance of neural networks with deeper layers.

For this project we used two of ResNet architectures: ResNet18 and ResNet34. As a result of the training and testing, ResNet18 outperforms ResNet34, so we will spend more time on explaining ResNet18. Besides, ResNet-18 is popular due to its balance of performance and computational efficiency.

- A 7x7 convolutional layer with 64 filters and a stride of 2.
- A 3x3 max pooling layer with a stride of 2.
- A series of 4 residual blocks, each containing two 3x3 convolutional layers.

5.2.1 ResNet18

In the following subsections, we will introduce the implementation details (hyperparameters, architecture, etc.) for the ResNet18 model we used. The following is the hyperparameters we used and

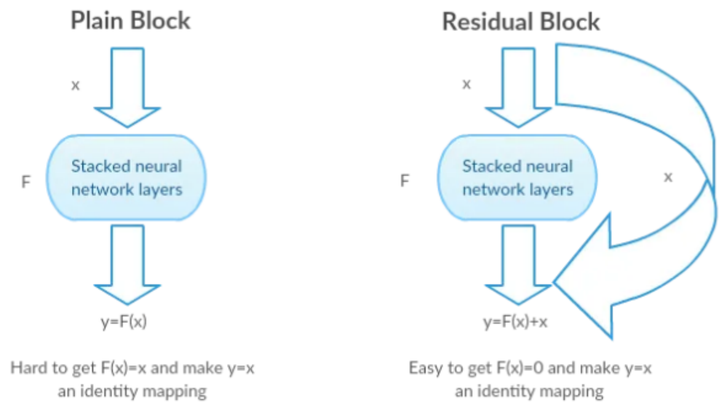
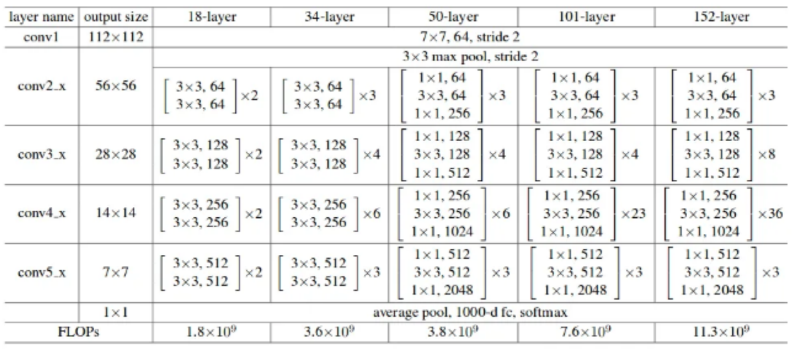

Plain Block
x
F
Stacked neural network layers
y=F(x)
Hard to get F(x)=x and make y=x an identity mapping
Residual Block
x
F
Stacked neural network layers
x
y=F(x)+x
Easy to get F(x)=0 and make y=x an identity mapping

Figure 8: ResNet Residual Block, source: here



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64 3×3, 64] ×2	[3×3, 64 3×3, 64] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3
conv3_x	28×28	[3×3, 128 3×3, 128] ×2	[3×3, 128 3×3, 128] ×4	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 128 3×3, 128 1×1, 512] ×8
conv4_x	14×14	[3×3, 256 3×3, 256] ×2	[3×3, 256 3×3, 256] ×6	[1×1, 256 3×3, 256 1×1, 1024] ×6	[1×1, 256 3×3, 256 1×1, 1024] ×23	[1×1, 256 3×3, 256 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512 3×3, 512] ×2	[3×3, 512 3×3, 512] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3
	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

Figure 9: ResNet Architecture, source: here

also we go over each term one by one.

- Model Architecture: ResNet18
- Loss Function: Cross Entropy Loss.
- Regularization: L2 with weight decay 0.0001
- Learning Rate: [0.01, 0.005, 0.0001, 0.00001]
- Number of Epochs: [5, 10, 15, 25]

5.2.2 ResNet34

In the following subsections, we introduce the implementation details (hyperparameters, architecture, and etc.) for the ResNet34 model we used. Which is very similar to the ResNet16 we used since both ResNet models are quite similar. The following are the hyperparameters we used and also we go over each terms one by one.

- Model Architecture: ResNet34

- Loss Function: Cross Entropy Loss.
- Regularization: L2 with weight decay 0.00001
- Learning Rate: [0.01, 0.005, 0.0001, 0.00001]
- Number of Epochs: [5, 10, 15, 25]

Cross Entropy loss function is a measure from two probability distributions, in which the Entropy of a random variable X is the level of uncertainty inherent in the variable's possible outcome. Cross-entropy loss is used when adjusting model weights during training which aims to minimize the loss.

$$L_{\text{CE}} = -\sum_{i=1}^{n} t_i \log(p_i), \quad \text{for n classes,}$$

where t_i is the truth label and p_i is the Softmax probability for the i^{th} class.

Figure 10: Cross Entropy, source: here

The reason we choose Adam is that it is a replacement optimization algorithm for stochastic gradient descent for training deep learning models. Adam combines the best of the AdaGrad and RMSProp algorithms to provide an optimization algorithm. So that it can handle sparse gradients on noisy datasets.

6 Discussion and Analysis

6.1 CNN

CNN Test Result						
Trial	Conv Layer	Epochs	Learning Rate	Training Accuracy	Test Accuracy Original Set	Test Accuracy New Set 1
1	2	10	0.001 (weight decay=0.0001)	68.22%	75%	0
2	4	10	0.001 (weight decay=0.0001)	98.61%	100%	34.62%
3	4	10	0.0005 (weight decay=0.0001)	98.44%	100%	38.46%
4	4	10	0.001	98.87%	100%	30.77%
5	4	5	0.001 (weight decay=0.0001)	96.46%	96.43%	26.92%

Genreally speaking, the CNN model performs well in terms of training accuracy and test accuracy on the original test set. However, it performs poorly on the new test set which we generated. There are several possible reasons:

- Model overfitted during training and probably relied on background for classification, instead of using hand gestures.

- There are discrepancies between the train set images and the test images we collected. In general, the train set is not inclusive enough as there are different forms of individual letters, specifically E,M,N,G,T,B,Z.[7]
- The new test images are not well-processed and have a lot of noise in them, even if the noise is almost invisible to human eyes.

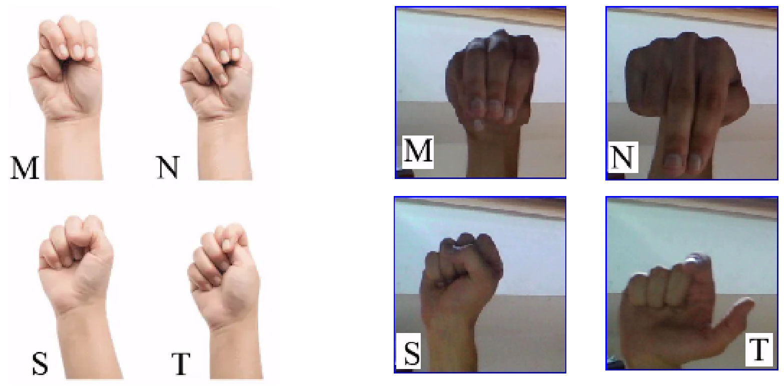


Figure 11: Example Images of the new test set (left) and the training set (right). The actual images do not have labels on them.

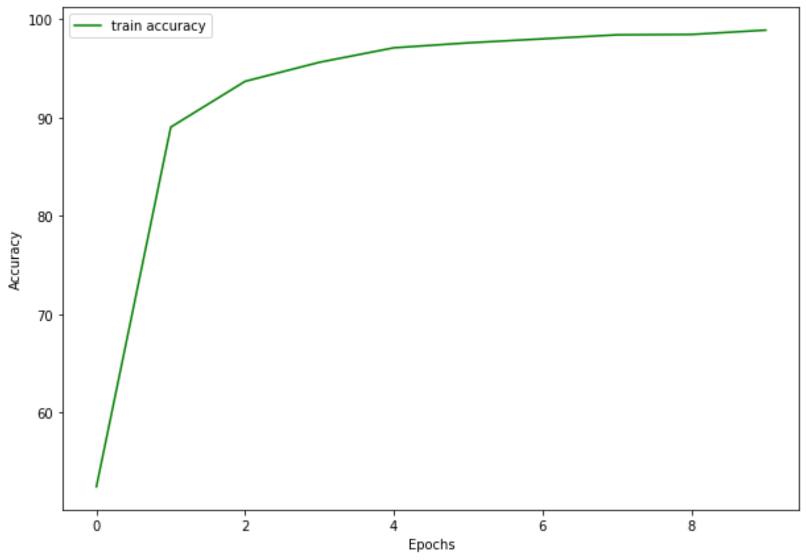


Figure 12: Accuracy Plot for CNN Trial 2

From the results, we can see how a model which performed quite well in the original test set can have low accuracy in some other test sets. This an engineering challenge that we need to find a balance between training accuracy and model generality. Trying to improve the performance, we have tried the following

- Reducing the learning rate as the training accuracy is above 90% at epoch 3 (shown in the figure 12), so the learning rate is more than sufficient. The comparison between trial 2 and 3 shows reducing the learning rate improves the performance of the model.

- Verifying that adding regularization actually reduces the over-fitting. The comparison between trial 2 and 4 shows that adding regularization improves the performance of the model.
- Trying to end the epochs early to preserve model generality and as the training accuracy is already high enough. The result is insignificant.

Another engineering challenge worth mentioning is that although the test accuracy can be improved simply by adding more diverse data into the training set, we need to consider the time to train the model and the chance that the model is not converging.

6.2 ResNet

6.2.1 ResNet18

During the training phase, using the ResNet18 model with the hyperparameter discussed in the section above, we can achieve 96.12%. Specifically, the learning rate used for this training is 0.001.

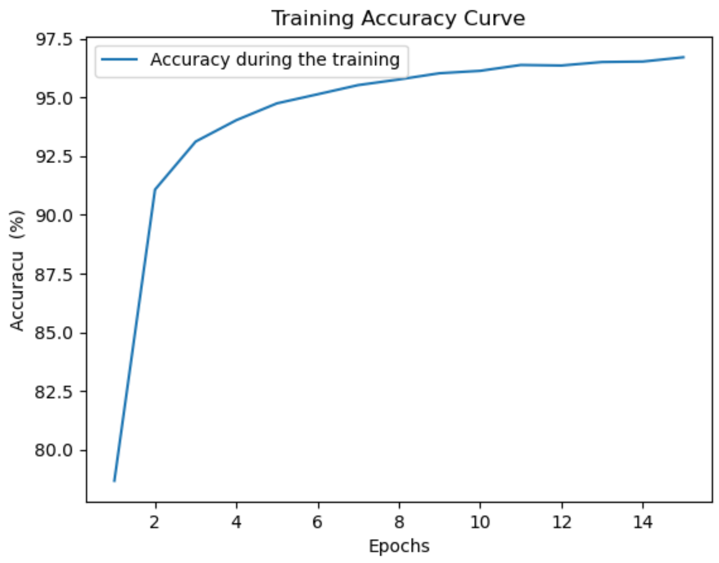


Figure 13: Accuracy Plot

During the testing phase, we can achieve 96.15% accuracy with the original test sets from Kaggle. Although, there are some cases the ResNet18 model can achieve 100 % accuracy, the worst case ever reported is 96.15%. Specifically, the learning rate used for the testing is 0.001. Therefore, in this report, I used the worst case as an example to plot a confusion matrix shown below. From the confusion matrix, we can tell the ResNet18 model incorrectly classify class 8 into class 25, which are I and Z respectively. As human point of view, I and Z represented in ASL is quite similar, I used little finger while Z used index finger. Therefore, it make some senses the ResNet18 might incorrectly classify image of I and image of Z.

As part of the requirement, we also tested some new data sets, which we discussed in section 4. Both newly added test sets do not perform well. The best result we see are 10.34% and 17.24% respectively. The ResNet18 model acts more like a random guessing. The reason that the accuracy is such low is the model is overfitting and it has a very high bias to the training sets. Therefore, it performs very badly in the new data sets.

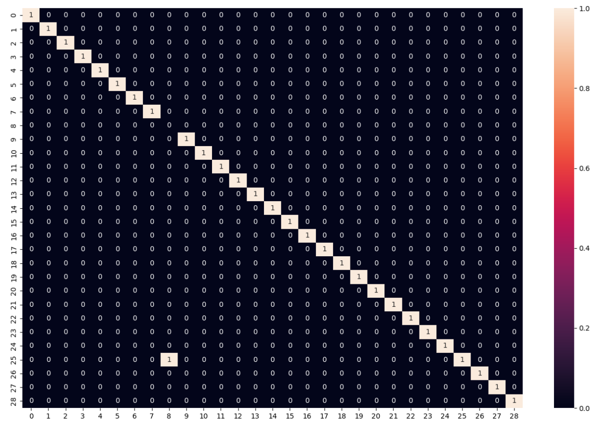


Figure 14: Accuracy Plot

6.2.2 ResNet34

Beyond the ResNet16, we also train and test the ResNet34 model. During the training phase, using the ResNet34 model with the hyperparameter discussed in the section above, we can achieve 95.72%. Specifically, the learning rate used for this training is 0.001.

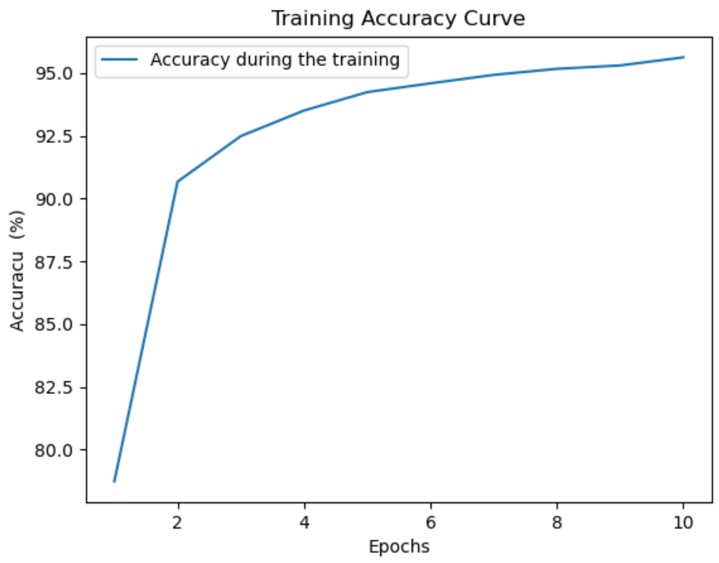


Figure 15: Accuracy Plot

During the testing phase, we can achieve 95.72% accuracy with the original test sets from Kaggle. Mostly, the ResNet34 model can achieve 100 % accuracy, the worst case ever reported is also 96.15%. As mentioned above, in the following report, I choose an example in which ACE all the tests to plot a confusion matrix shown below. From the confusion matrix, we can tell the ResNet34 model correctly classify all images.

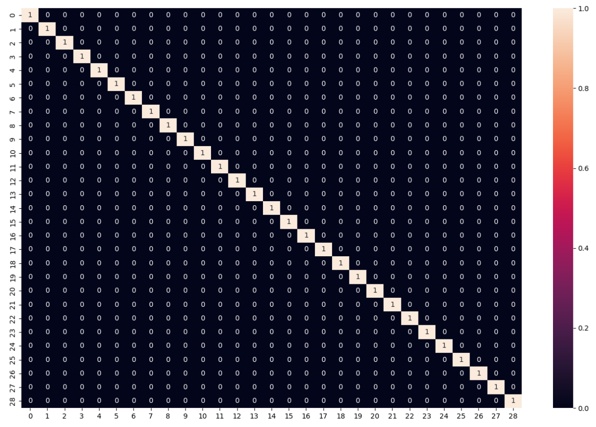


Figure 16: Accuracy Plot

As part of the requirement, we also tested some new data sets, which we discussed in section 4. Both newly added test sets do not perform well. The best result we see are 3.57% and 10.34% respectively. The ResNet34 model performs somehow even worse than ResNet18. The reason that the accuracy is such low is the model is overfitting and it has a very high bias to the training sets. Therefore, it performs very badly to the new data sets.

7 Conclusion

The project is to design and implement various deep learning models to train and test given ASL data sets from Kaggle and additional self-created data sets. In this project, we implemented a few different CNN and ResNet deep learning to detect ASL images.

Both ResNet models (ResNet18 and ResNet34) perform very similarly. They have very high accuracy to detect the training and testing sets from Kaggle but perform very poorly in testing with new self-created data sets.

CNN models also have high accuracy for Kaggle training and testing sets, and it can achieve modest results (≈38%) for the new test set. We have tried numerous ways to reduce the over-fitting, but as training set is not very inclusive, we might need a larger training set to resolve the over-fitting issue completely.

For future work, we could create a new training and testing large data sets. Then, we could test our both CNN and ResNet deep learning models comprehensively. We should add more diversity into the training set so that the model is less affected from backgrounds and light levels. We are also include gestures from multiple people as we are aware that ASL alphabet is not fully standardized, and there are multiple ways to spell the same letter.

As ASL is a sign language, and is not 'spoken' by spelling letter by letter, we think it's very interesting to identify more ASL symbols, and add gesture recognition for video input so that we can understand the entire sentence. In addition, ASL users' motions are continuous while they 'speak', so the time dimension has to be taken into consideration. We can either use the previous collected symbols to put more weights on next symbols, or using state-of-the-art dynamic gesture recognition to interpret the sentence all together. We are exited to explore this academic area.

Along the way, this project gave us a review over the course subject and fortify the our knowledge foundation in machine learning. We encountered some unexpected difficulties, such as how to generate a test set, but we successfully solved them. Before starting the project, we always aimed for the highest training accuracy, but now we have a better idea of over-fitting and how do deal with it.

In the end, we are curious about the data preparation and the boundary of potentials of CNN and its related models: it seems that they are capable of capturing the most subtle patterns, but they are also highly susceptible to noises that our naked eye don't really pay attention to. Sometime it seems more important to better prepare the data before we send them into the model rather than to build a more sophisticated model to deal with noises. However, the world is inherently noisy, so we should always be ready to balance the performances and make engineering trade-offs.

8 Code Repository

ASL Project Code Repository

References

[1] American sign language alphabet recognition. `https://medium.com/mlearning-ai/american-sign-language-alphabet-recognition-ec286915df12`. Accessed: 2023-4-14.

[2] American sign language (asl) alphabet (abc) poster. `https://www.gerardaflaguecollection.com/products/american-sign-language-asl-alphabet-abc-poster.html`.

[3] Basic cnn architecture: Explaining 5 layers of convolutional neural network. `https://www.upgrad.com/blog/basic-cnn-architecture/`.

[4] Classifying the asl alphabet using machine learning. `https://medium.com/quick-code/classifying-the-asl-alphabet-using-machine-learning-83037ac463e6`. Accessed: 2023-4-14.

[5] A day in our shoes. `https://adayinourshoes.com/asl-fingerspelling/`.

[6] Quick statistics about hearing. `https://www.nidcd.nih.gov/health/statistics/quick-statistics-hearing`.

[7] Sign language: Fingerspelling. `https://www.lifeprint.com/asl101/pages-layout/fingerspelling.htm`.

[8] Garcia B. and Viesca S. A. Real-time american sign language recognition with convolutional neural networks., 2016.

[9] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016.

[10] Nikhil Kasukurthi, Brij Rokad, Shiv Bidani, and Dr. Aju Dennisan. American sign language alphabet recognition using deep learning, 2019.

[11] Gongfa Li, Heng Tang, Ying Sun, Jianyi Kong, Guozhang Jiang, Du Jiang, Bo Tao, Shuang Xu, and Honghai Liu. Hand gesture recognition based on convolution neural network. *Cluster Computing*, 22:2719–2729, 2019.

[12] Ross Mitchell, Travas Young, Bellamie Bachleda, and Michael Karchmer. How many people use asl in the united states? why estimates need updating. *Sign Language Studies*, 6, 03 2006.

[13] Yanze Wang and Junyong Ye. Tmf: Temporal motion and fusion for action recognition. *Computer Vision and Image Understanding*, 213:103304, 2021.

[14] Jimin Yu, Maowei Qin, and Shangbo Zhou. Dynamic gesture recognition based on 2d convolutional neural network and feature fusion. *Scientific Reports*, 12(1):4345, 2022.