American Sign Language Recognition using Deep Learning

Chandhini Grandhi
A53272378
cgrandhi@eng.ucsd.edu

Sean Liu
A92094374
sel118@eng.ucsd.edu

Divyank Rahoria
A53284395
drahoria@eng.ucsd.edu

Abstract

This paper explores the application of deep learning to the task of multi-class classification of American Sign Language (ASL). We aim to classify every image in the ASL dataset to one of 29 classes. We develop a fully convolutional neural network (CNN) to achieve this on our ASL dataset. With this as our baseline, we look into how different CNNs would potentially improve our performance. We trained our model on a VGG16 network from scratch and used pretrained weights to see how transfer learning affects the performance. Also, we wanted to train the model with deeper and wider networks to see the effects on performance, and for this, we trained with InceptionNet and ResNet50.

1 Introduction

American Sign Language (ASL) is a form of communication for people with speech impairment, predominantly used for deaf communities in the United States and most of Anglophone Canada. The problem with ASL is that it is a challenge for non-sign language speakers to communicate with sign-language speakers. Using deep learning, we tackle this problem using the ASL dataset available here

This dataset has images of hand gestures of 29 classes. The details of the dataset is explained in Section 3

2 Related Work

There have been several approaches to address the issue of translating sign language into text using different deep learning models in recent years. For example, (5) uses Inception, Convolutional Neural Network (CNN) for recognizing spatial features and Recurrent Neural Network (RNN) to train on temporal features. (6) uses Convolutional Neural Network (CNN) to predicting Sign Language and achieved 95% accuracy. (8) uses Convolutional Neural Network (CNN) and recorded the weights and model for real-time prediction. (9) talks about the relevant features of the model, feature extraction and uses Artificial Neural Network (ANN) to classify signs. Finally, (10) proposed a system where they used AdaBoost and Haar-like classifiers to detect and translate ASL alphabets. We aim to get better accuracy than previous work and try out different architectures to compare the performance of the models.

3 Dataset

The data set contains 87,000 images with 29 classes, where 26 are for the letters A-Z and 3 classes are for SPACE, DELETE and NOTHING. These 3 classes are very helpful in real time applications and classification. To reduce computation time, we decided to choose 21750 images, or 750 images per class, to be used for our models.

Figure 1: Input Data Visualization

We preprocessed the data by resizing to 200x200 pixels, using a random shuffle and normalization of the data. For Inception Net model, we scaled down the resolution further to 50x50 for faster computation and lower memory consumption. We split those images into three sets of training, validation and test set. Training set has 13920 images, validation set has 3480 images and test set has 4350 images. The visualization of input data can be seen in figure 1.

4 Methods

4.1 Baseline CNN

CNNs are a deep learning algorithm that takes in an input image and assigns importance to various aspects in the image. It captures the spatial and

temporal dependencies of the image. Each CNN layer learn filters of increasing complexity. The first layers learn basic feature detection filters like edges, corners. The middle layer learn filters that detect parts of objects. The last layers have higher representations- they learn to recognize full objects in different shapes and positions. The CNN used here serves as a basis for some of the later models such as VGG16 and InceptionNet.

4.2 VGG16

The next convolutional neural network that we have attempted to use is VGG16. Essentially, this network is even deeper than the baseline CNN network from section 4.1 because it stacks convolutional layers on top of each other. In other words, instead of just having a single convolutional layer before a max-pooling layer, consecutive convolutional layers are used before a max-pooling layers. In addition, by using multiple ReLu, or activation, units successively, the decision function is more discriminative(11). Figure 2 shows the VGG16 architecture in even further detail(1).

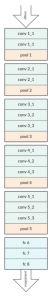

Figure 2: VGG-16 Architecture

4.3 InceptionNet

Furthermore, InceptionNet is a CNN that uses multiple sized kernels and concatenation. Instead of going deeper than the baseline CNN like VGG16 from section 4.2 does by stacking convolutional layers, it increases the number of layers by widening the network, parallelizing convolutional layers at the same stage. An example of the network used is shown in Figure 3. Essentially, at the same stage of the network, convolutional layers with filter sizes of 1x1, 3x3, and 5x5 are all used, and the resulting layers are concatenated depth-wise. Doing this allows capture of salient features at multiple levels, as both global features distributed over a large area of the image and area-specific features distributed across the overall image can be simultaneously detected and accounted for(4).

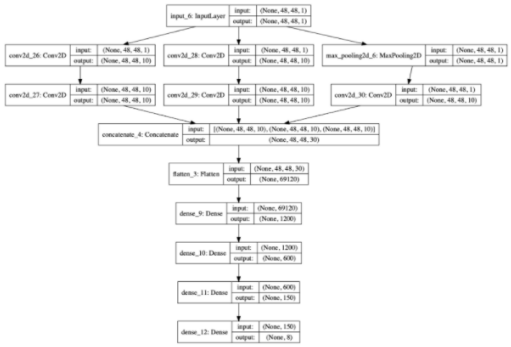

Figure 3: InceptionNet Architecture

4.4 ResNet50

ResNet(3) is one of the most powerful deep neural networks that achieves great performance by using excellent generalization in recognition tasks, and ResNet50 is a variant that is 50-layers deep. ResNet uses batch normalization that adjusts the input layer to increase the performance. This helps mitigate the covariate shift problem. ResNet is similar to other networks which have convolution, pooling, activation and fully-connected layers stacked one over the other. The only difference is construction of identity connection between the layers as shown in figure 4. This block of layers with identity connection is known as residual block, where identity connection is the curved arrow originating from the input and sinking to the end of the residual block. Identity Connection helps protect the network from vanishing gradient problem.

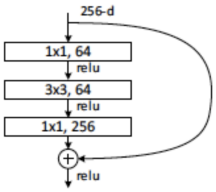

Figure 4: Residual block

4.5 Loss Function

Categorical cross-entropy loss is used for VGG-16, InceptionNet. It uses one-hot encoding while sparse categorical cross entropy loss is used in CNN, ResNet50 which has an output in the form of labels. Both the losses are used for multi-class classification. it compares the distribution of all predictions with the true distribution. The loss function is given in the equation 1.

$$L = -\sum_{n}\sum_{k=1}^{c} t_k^n \ln y_k^n \tag{1}$$

Where c are the classes, n are all the images t_k^n is the target for images n and y_k^n is the predicted softmax output for the image.

4.6 Optimizer

4.6.1 Adam

Adam optimizer is used in the base CNN, InceptionNet, and ResNet50 models. Adam provides an optimization algorithm that can handle sparse gradients on noisy problems. We chose Adam because it is computationally effective and requires little memory space.

4.6.2 SGD

In addition to using the Adam optimizer, we also used SGD, or Stochastic Gradient Descent, for both the VGG16 and VGG16-pretrained models. This is based on a study in (12), which found that for VGG specifically, it was found that SGD actually worked much better in reducing training and testing error, and Adam actually performed worse than any optimizer tried. Therefore, using SGD as an optimizer, with a learning rate of 0.001 would perform better.

5 Experiments

All models were built using Relu activation functions for all layers and a softmax layer at the output to predict the classes.

5.1 Experiment 1: Baseline CNN

We experimented by building a deep CNN for the purpose of classification (2). The overview of our architecture is we have Convolutional layers to learn the features of the images, each followed by Max Pool layers to reduce the number of parameters to learn and also to help in translation invariance. We have then used BacthNormalization layers along with Dropout to reduce the overfitting and to speed up the computation. We experimented with architectures and found out the current architecture gave us the best performance. The architecture is also shown in figure 5.

Layer	Output Shape	Param
Conv	(None,200,200,16)	448
Maxpool	(None,66,66,16)	0
Conv	(None,64,64,32)	4640
Maxpool	(None,21,21,32)	0
Conv	(None,19,19,64)	18496
Maxpool	(None,6,6,64)	0
BatchNorm	(None,6,6,64)	256
Flatten	(None,2304)	0
Dense	(None,1000)	2305000
Dropout(0.4).	(None,1000)	0
BatchNorm	(None,1000)	4000
Dense	(None,500)	500500
Dropout(0.4).	(None,500)	0
BatchNorm	(None,500)	2000
Dense(Softmax)	(None,29)	14529

Figure 5: CNN Architecture

5.2 Experiment 2: VGG16

For VGG16, the model was built from scratch, modified, and trained exclusively using our dataset. However, a key drawback with VGG is that it is slow to train, as training a single epoch took over a minute. With that in mind, a VGG16 model via transfer learning with pretrained weights based on Imagenet was also implemented (7). The overall network looks similar to that of Figure 2, except with BatchNormalization and Dropout layers added in a pattern similar to that of the baseline CNN.

5.3 Experiment 3: InceptionNet

Next, we modified InceptionNet by adding BatchNormalization layers after each dense layer and concatenation block to try and prevent overfitting, and trained from scratch, once again in similar pattern as the baseline CNN. Image sizes were also reduced further to 50x50 to lower memory consumption, and 1x1 bottleneck layers were introduced to limit the amount of feature maps generated. It is clear that with InceptionNet, training will be much faster especially compared to the base CNN or VGG models, with a single epoch taking just 7 seconds. Overall, the network will be very similar to that in Figure 3.

5.4 Experiment 4: ResNet50

We experimented by building a modified ResNet50 architecture(3) from scratch and trained exclusively using our dataset. The identity block we used has 3 components and a final step. These components are built by convolution layers, Batch normalization and activation layers as shown below.

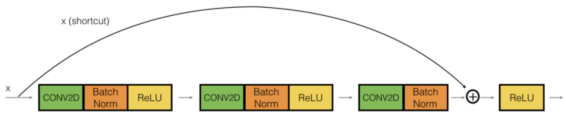

Figure 6: Identity Block Architecture

The convolutional block has similar architecture and a shortcut path that eventually gets added to main path as shown below.

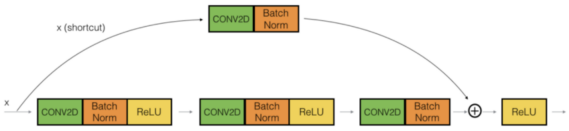

Figure 7: Convolutional Block Architecture

The overview of the architecture is shown in figure 8.

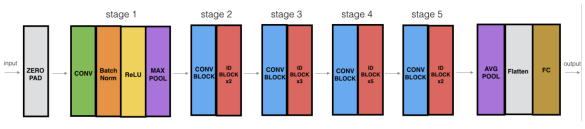

Figure 8: Overall ResNet50 Architecture

6 Results

All models were trained for 70 epochs to maintain consistency.

6.1 Experiment 1: Baseline CNN

Results obtained from training the baseline CNN from scratch are presented below. The test accuracy obtained is 98.02%

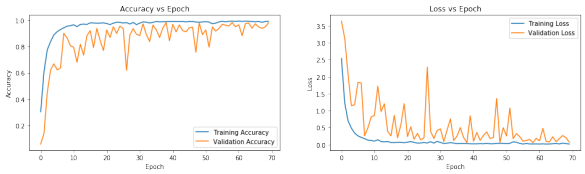

Figure 9: Accuracy and Loss vs Epochs for CNN

6.2 Experiment 2: VGG16

Results obtained from training VGG16 from scratch and through transfer learning are listed below. The VGG16 model with transfer learning trained only over 15 epochs. Final testing accuracy achieved for the VGG16 model trained from scratch was 99.74%, while for the transfer learning model, it was 99.84%.

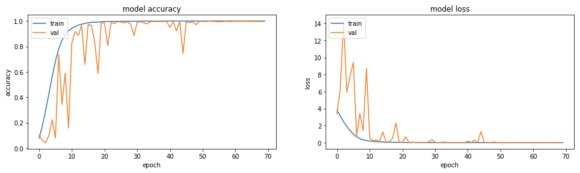

Figure 10: Accuracy and Loss vs Epochs for VGG16 (trained from scratch)

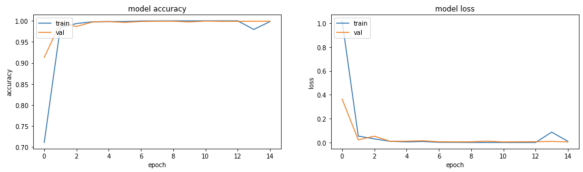

Figure 11: Accuracy and Loss vs Epochs for VGG16 (with pretrained weights)

6.3 Experiment 3: InceptionNet

Next, results obtained from training InceptionNet are shown below. The final testing accuracy achieved was 96.76%.

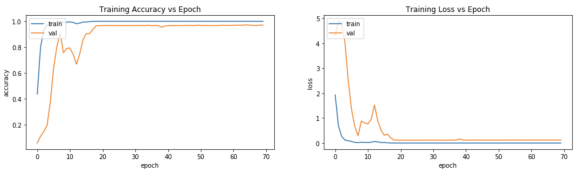

Figure 12: Accuracy and Loss vs Epochs for InceptionNet

6.4 Experiment 4: ResNet50

The results obtained from training the ResNet50 are presented below. The final test accuracy achieved was 99.88%, highest among all the models used in this paper.

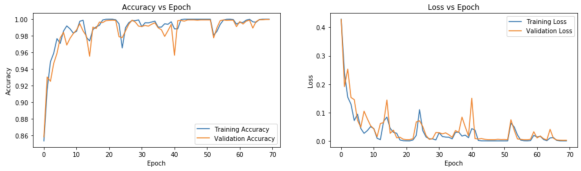

Figure 13: Accuracy and Loss vs Epochs for ResNet50

6.5 Visualization of predictions

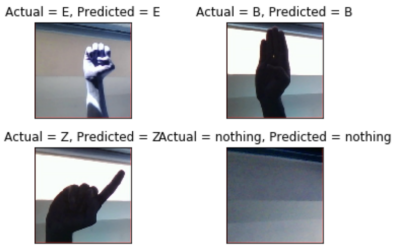

Figure 14: Visualization of Predictions

To summarize the results, here is the comparative table for all the tested models over test set in Table 1. Results for all models are generally very good.

Model	Test Accuracy
CNN	98.02%
VGG16 Scratch	99.74%
VGG16 Pretrained	99.84%
InceptionNet	96.76%
ResNet50	99.88%

Table 1: Model Comparison

7 Conclusion

The aim of this project was to find a model with highest accuracy for the task of multi-class classification of American Sign Language. In this paper, we compared four different CNN models for hand gesture recognition for ASL. Overall, we have found that ResNet50 achieved the best results training from scratch, while using pretrained weights for VGG16 proved the effectiveness of transfer learning. These models can be very effective for the purpose of ASL translation, and for the future, we hope these classification networks can be used, built on further, and even combined with temporal data and recurrent neural networks to learn sequences of words and sentences.

References

[1] Convolutional neural network architecture: Forging pathways to the future.

[2] Data pre-processing and cnn tutorial. https://keras.io/examples/mnist_cnn/ ,Kagglekernels.

[3] Residual networks tutorial. https://pylessons.com/Keras-ResNet-tutorial/.

[4] A. Anwar. Difference between alexnet, vggnet, resnet and inception, May 2020.

[5] 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.

[6] 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.

[7] V. Kommineni. How to use transfer learning for sign language recognition, Mar 2019.

[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] V. N. Truong, C.-K. Yang, and Q.-V. Tran. A translator for american sign language to text and speech. In *2016 IEEE 5th Global Conference on Consumer Electronics*, pages 1–2. IEEE, 2016.

[11] J. Wei. Vgg neural networks: The next step after alexnet, Jul 2019.

[12] A. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht. The marginal value of adaptive gradient methods in machine learning. 05 2017.

Individual Contributions

Chandhini

Worked on preprocessing the data, visualizing images, results and to obtain loss and accuracy curves so that it is consistent for all the networks. Worked on implementation and gathering results from CNN model. In the report, contributed to abstract, introduction, CNN section.

Divyank

Worked on literature survey and listed out all related previous work in this domain to come up with some initial possible architecture ideas.Worked on implementing ResNet50 model. In the report, contributed to abstract, Related work, Dataset, ResNet50, Conclusion and Reference section.

Sean

Worked on literature survey, building training, validation, and test sets/split, and loss and accuracy curves. Worked on implementation of VGG16 and InceptionNet. In the report, contributed to abstract, VGG16, InceptionNet, Optimizer, Conclusion and Reference section.

Replies to Critical reviews

Group 36: Group 36 offered a lot of suggestions on how we could improve, and we have taken most of this into account.
1. *It would be better to refer your model as (proposed implementation) because all other methods also use convolution neural networks*
We realized that our CNN label may have been a little misleading, as it was not a proposed method as it was a baseline CNN method that we use when comparing and extending it to deeper and wider networks such as VGG and InceptionNet. This has since been reflected in the paper.
2. *How your proposed method is better than VGG16 -Pretrained and Resnet50 because the later ones seem to give reasonable better accuracy than yours?*
Once again, because the CNN label as a proposed implementation is a little misleading, this has been reflected in our final paper as well. It would be expected for Resnet50 or VGG16-Pretrained to perform better given they are deeper networks or have pretrained weights.
3. *Can you explain the reason why your model has failed to achieve more than 90 percent accuracy?*
While it is not our model, we had realized that our lack of usage for a validation set allowed the model to overfit for the training data. Once we added a validation set before using a test set, all models performed much better in terms of testing accuracy.
4. *Can you please explain why did you only these architectures for this problem and comparison? Why not others?*
Overall we chose these architectures because of a combination of factors such as runtime, depth, width, and related research. For example, for Resnet, we were aware that this was one of the deepest networks with plausibly high accuracy. Likewise, much of the literature found had experimented with InceptionNet.
5. *It would be better to plot all curves in one figure to show comparison.*
This has been reflected in the final report.
6. *As you have computed training time for each method, it would be better to summarize your comparison in one plot or histogram.*
We have taken this into account, but decided for our final report that including another plot would take too much space.
7. *Please clarify did you use any data preprocessing techniques to improve the results? If not, please explain the reason. If yes, please explain a little bit about it.*
For the most part, data preprocessing included normalization of the data and scaling and downsampling the data, and this has been reflected in the report.

Group 66: Thank you for your feedback and indeed pre-trained VGG gave us a really good accuracy and the training was faster for it as well. This shows the importance of transfer learning on a pretrained model.

Group 59: Thanks for your feedback. Our models did not overfit, to verify this we further split our training set into train and validation and re-trained our models and it performed equally good. Also, the test accuracy that was shown as 80% in the demo was because for the purpose of demo we trained the model only for 50 epochs.