American Sign Language Recognition using a CNN (ICML 2022)

Malcolm Thorpe [*1] **Nishi Ajmera** [*1 2] **Sinclair Fuh** [2]

Abstract

American Sign Language (ASL) is the predominant sign language for the Deaf communities in both the United States and most of Anglophone Canada. ASL is an organized type of visual language that uses both manual and nonmanual features to make expressions. We want to create a Machine Learning model that can reliably identify ASL. In this paper, we will discuss the approach that will be used to accomplish this task. We will be using a CNN (Convolutional Neural Network) that will be trained by breaking down images that contain ASL gestures and it will learn the key differences between the different letters in the images so it can accurately predict various ASL gestures after being trained.

1. Problem Description and Motivation

Reliably identifying ASL can be seen as a litmus test for the quality of an image-identification model: it's a common real-world use for image classification models that requires them to be able to consider subtle differences between images. For example, both the poses for "A" and "E" are balled-up fists. The only difference between the two is whether or not the thumb is held to the side, with "A" having its thumb to the side and "E" having its thumb held at the front of the fist. To reliably identify sign language, image recognition models need to be able to identify these sorts of details in images. Image recognition models that only consider factors such as the outline of an image or the color of each pixel would have a limited ability to articulate these details, but CNNs have been reliably used to identify poses in sign language before. We want to implement a CNN that demonstrates the capability to recognize such minute differences between images.

2. Explored Questions

The datasets we use include static, full-color images of separate hand gestures used in ASL. This raises the question of how we'll identify which parts of the image are the hand and which aren't. While this might be a question of how we design our features, it could also be a question of how we preprocess our images before putting them into the neural network. It might be more effective to convert the image into a black and white outline of the hand, as opposed to feeding our neural network the raw image of the hand. Our project dips into the realm of image processing as a result, and it's something our group will explore as we develop our CNN.

3. Related Work

A CNN model was used by Pigou in 2014 when he researched sign language, where he used the CLAP14 database. The database was made available by Microsoft Kinect. Preprocessing of the database was done by cropping the highest hand and the upper body, further thresholding was performed to reduce the noise in-depth maps and background removal. The database consisted of 6600 videos from which he used the first 4600 for the training and the rest 2000 for the validation. The proposed model by Pigou consists of 6 layers which include input and output. A ReLU activation was used and the model performed at 91% accuracy with an 8% error rate (Pigou, Dieleman, Kindermans, et al., 2014).

A 3D Convolutional Neural Network was used by J Huang in 2015 in which he did research about sign language but didn't mention which county or standard. He used his dataset which was made using Microsoft Kinect. He hasn't done any pre-processing with the dataset. The model consists of 8 layers including input and output. The model had 4 layers of a 3D convolutional network and got an accuracy of 94.2% with an error rate of 5.8%(Suharjito, Gunawan, et al., 2018).

4. Proposed Solution

4.1. Analyzing the Data

Before implementing the various CNN models we analyzed the American Sign Language dataset that will be used throughout the project implementation for training, development, and testing. The first thing we did was look throughout the training set and identify how the distribution

of the letters looked throughout the dataset.

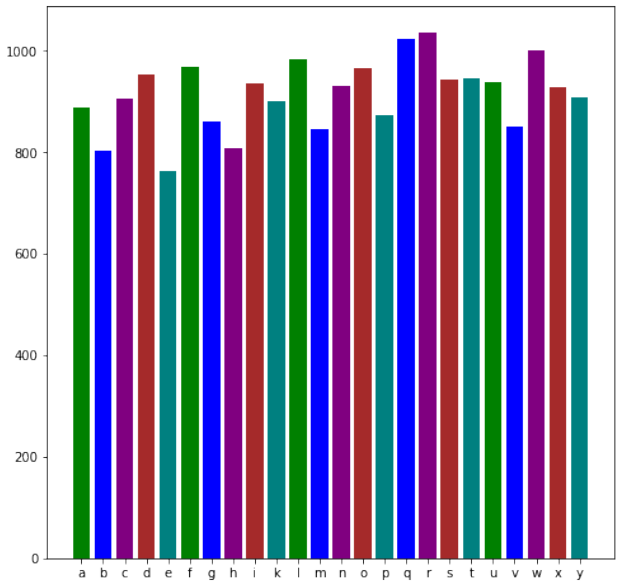

Figure 1. Graph showing the Distribution of The ASL hand gestures in the dataset excludes J and Z

The distribution of the dataset shows that the different labels that are represented are fairly and evenly distributed throughout the data, with the gesture "r" occurring the most. Once this was identified we decided on how to split the full training set so that part of it could be used for developmental evaluation. We have 27,455 images for the training dataset which we will split to be used for the developer dataset and we have 7172 images for the test dataset. When splitting the full training set we took the first 20% of the data and used the it for the development data set. The remaining 80% of the training set is used for the the training of the CNN models.

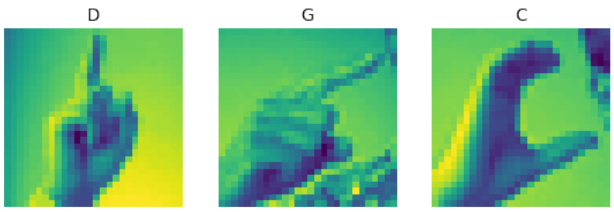

Figure 2. Example of what an image in the dataset is interpreted as when being passed into the models

In Figure 2 shows an example of how a single image of the data set can be interpreted. Here the images are of 28 x 28 pixels. Being able to look at the image and break it down helps you to identify what the best kernel size maybe for the model to properly learn the image that is provided.

4.2. Design 1: SDCNN

My proposed Convolutional Neural Network has two layers, which consist of a convolutional layer, followed by a Max Pooling layer and a Dropout layer. I observed how the varying kernel sizes are affecting the performance of the CNN models while keeping the architecture fixed. Furthermore, I analyzed the accuracy and loss in the different layers.

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 1, 4, 32)	43936
max_pooling2d_2 (MaxPooling 2D)	(None, 1, 2, 32)	0
dropout_2 (Dropout)	(None, 1, 2, 32)	0
conv2d_3 (Conv2D)	(None, 1, 1, 128)	200832
max_pooling2d_3 (MaxPooling 2D)	(None, 1, 1, 128)	0
dropout_3 (Dropout)	(None, 1, 1, 128)	0
flatten_1 (Flatten)	(None, 128)	0
dense_1 (Dense)	(None, 64)	8256
dense_2 (Dense)	(None, 24)	1560

Total params: 254,584
Trainable params: 254,584
Non-trainable params: 0

Figure 3. CNN model summary of the CNN model having (4,4) kernel size, two Convolutional layer followed by one Max Pooling layer and a Dropout layer each

4.3. Design 2: American Sign Language CNN

The American Sign Lanuguge CNN (ASLCNN) is the convolutional neural network machine learning model that has been implemented in the second design. This CNN contains three convolutional blocks and one fully connected layer that have been implemented in the model. Within the convolutional block there is various layers such as batch normalization, rectified linear unit, max pooling, and dropout. The batch normilzation and dropout layer were implement to prevent overfitting on the training data. The dropout is set to 0.5 in the first layer and to 0.8 in the second layer increase makes it harder to overfit as the image features are passed deeper into the model.

4.4. Design 3: LeNet-5 applied to ASL

The original creator of the MNIST dataset also proposed a CNN able to achieve over 99% accuracy on it, titled LeNet-5. [4] Both LeNet-5's role in inspiring modern CNN designs such as AlexNet and the MNIST dataset's similarity

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 16, 28, 28]             160
       BatchNorm2d-2          [-1, 16, 28, 28]              32
              ReLU-3          [-1, 16, 28, 28]               0
           Dropout-4          [-1, 16, 28, 28]               0
            Conv2d-5          [-1, 32, 28, 28]           4,640
       BatchNorm2d-6          [-1, 32, 28, 28]              64
              ReLU-7          [-1, 32, 28, 28]               0
         MaxPool2d-8          [-1, 32, 14, 14]               0
           Dropout-9          [-1, 32, 14, 14]               0
           Conv2d-10          [-1, 64, 14, 14]          18,496
      BatchNorm2d-11          [-1, 64, 14, 14]             128
             ReLU-12          [-1, 64, 14, 14]               0
        MaxPool2d-13            [-1, 64, 7, 7]               0
          Flatten-14                [-1, 3136]               0
           Linear-15                  [-1, 25]          78,425
================================================================
Total params: 101,945
Trainable params: 101,945
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 1.39
Params size (MB): 0.39
Estimated Total Size (MB): 1.78
----------------------------------------------------------------
```

Figure 4. CNN model summary of the CNN model having (1,1) kernel size, one convolutional layer followed by batch normilization layer, a ReLu layer, max pooling layer, and a dropout layer

to the datasets we used are the two primary motivators for implementing LeNet architecture to identify ASL poses. To elaborate, both our ASL dataset and the MNIST dataset are image datasets that represent each image as a greyscale, 28 by 28 image. Each of these greyscale images in both datasets are represented using a 28 by 28 array of integers, each a greyscale value that ranges from 0 to 255. In addition, LeNet-5 continuing relevancy to modern image classification tasks using MNIST-esque datasets [5] make LeNet-5 a promising choice for our task of ASL recognition.

5. Experimental Methodology

5.1. SDCNN

5.1.1. Effect of different the kernel sizes

For the used dataset, we have tries different kernel sizes for our model. It has been observed that kernel size of (4,4) gives the most accurate result among other kernel sizes for the SDCNN.

Kernel Size	Accuracy
4,4	91.62%
5,5	87.74%
6,6	88.84%
7,7	75.26%

Table 1. Accuracy for different kernel size

5.1.2. Effect of different convolutional layer

In the model the layers near to the input are used to classify the low-level feature whereas the layers which are far from input and deep in model are used to classify the high-level features. We have tried different number of layers and with that even the value of features in the layers makes a difference. If we keep same type of features and increase the number of convolutional layers the accuracy got higher. but for some kernel size some of the number of layers are not giving better accuracy.

Kernel Size	Number of layers	Accuracy
4,4	1	91.62%
4,4	2	92.13%
4,4	3	92.8%
4,4	4	85.46%
5,5	1	87.74%
5,5	2	89.9%
5,5	3	88.34%
5,5	4	85.23%
6,6	1	88.84%
6,6	2	89.04%
6,6	3	91.19%
6,6	4	82.10%
7,7	1	75.26%
7,7	2	67.29%

Table 2. Accuracy for different convolutional layer for each kernel size

5.2. ASLCNN

5.2.1. Effect of different dropout rates

Dropout is a simple way to prevent the CNN from overfitting. because the outputs are randomly sub sampled. when using no dropout on the ASLCNN the model was clearly overfitting to the training data because a perfect accuracy if 100% was obtained after the first epoch and so on. To optimize the CNN to be able to handle more types of datasets I had to implement dropout layers in the convolutional blocks to reduce the chances of this happening. When using a dropout of 0.5 on 1st CNN block only the training started to provided slightly much more realistic results and didn't score perfects scores on the first iterations of the the training, but later on after the first few epochs the issues continued. Then I decided to add dropout on the first 2 CNN blocks a dropout rate of 0.5 on the first CNN block and 0.8 on the second CNN block. This helped to completely get rid of the overfitting of the ASLCNN. In the analysis of the results The ALSCNN with the 2 dropout layer configuration will be analyzed from this model.

5.3. LeNet-5

In its original proposal, LeNet-5 contains 9 different layers (including the input, output layer and the dropout layer.) After the input layer come 3 convolutional layers and 2 max-pooling layers, both of which alternate between each other and include a dropout layer after the first max-pooling layer. Each convolutional layer utilizes a 5 by 5 kernel, and each max-pooling layer utilizes a 2 by 2 kernel. This is followed by a linear layer and the output layer. While the goal was to implement our model as closely to the original LeNet-5 as possible, the circumstances of our task demanded some parts be done differently. Notable differences in this implementation from the original include:

- Feature Regularization: In the original MNIST dataset used by LeCun et al., each image was that of a number drawn on a sheet of white paper. As a result, each image had a very clear foreground and background. This resulted in the greyscale arrays for each image including several low values, and a few high values that represented the area the number was drawn on. As a result, these greyscale arrays were regularized by setting each low-valued pixel indicative of the background to -0.1, and each high value indicating a pixel in the foreground where the digit was drawn on to 1.175.[4]

  In contrast, the ASL dataset we used had no clear distinction between the foreground and the background. Unlike the MNIST dataset, the background to each image wasn't pure white, and had a broad range of greyscale values that were often similar to the foreground in each image: the hand gesture representative of a letter in the ASL alphabet. This made implementing the same method of regularization difficult, since we can't conclusively define a cutoff value that indicates whether a pixel should be -0.1 or 1.175. Instead, 0-1 regularization was used in our implementation of LeNet-5: 0 being the smallest value and 255 being the highest.

- Layer C5: The final convolutional layer in LeNet-5's original implementation used a 5 by 5 kernel to produce a 1 by 1 feature map. In other words, the kernel at this step of the model covered its entire input to produce a single value. However, LeNet-5 also took 32 by 32 images as its input instead of the 28 by 28 images our model takes in. As a result, the input at this step in our model was a 4 by 4 array, smaller than the 5 by 5 kernel used in the original solution. This implementation uses a 4 by 4 kernel instead to achieve the same effect of covering the entire input to produce a single value.

- Dropoff methods: The original LeNet-5 dropped off specific, predetermiend layers after the first max pooling layer in its architecture. In order to introduce more variance in our dropout, this model implements a randomized dropout layer at the same point in our model with a dropout probability of 0.2.

- The output layer: LeNet-5 utilizes Gaussian connections in its output layer. However when experimenting with separate output layers, our implementation of LeNet-5 was able to achieve over 99% accuracy across all splits of our data using linear layers. For this specific task, we felt no need to further experiment with the output layer.

6. Analysis of Results

6.1. SDCNN

So as the result of above discussed parameters, we have selected to use 4,4 as kernel size with 2 convolutional layers. This model has proven to be the best SDCNN model for our selected data. From the model it can be seen that the model is most confident in predicting 'C', 'D', 'F', 'G', 'I', 'K', 'L', 'O', 'M' and least confident in predicting 'W. For this model the best accuracy is 92.13%

6.2. ASLCNN

One thing that can be noticed when using dropout layers is each time you retrain the model it may perform slightly better or worse because of the randomness of where the dropout will occur. The tables below show the analysis of one of the training models that was run.

Development Evaluation		
Epoch	Accuracy	Average Loss
1	92.8%	0.322053
2	98.5%	0.111240
3	99.1%	0.063767
4	99.8%	0.027786
5	99.77%	0.026653

The Development Evaluation table shows how the ASLCNN model performed after each of the epochs. In reality, when retraining the accuracy after an epoch can vary from 90-100% from what was gathered after training multiple models. The reason that accuracy is a good metric used to verify how the model is performing is that the dataset has a good and fairly distributed amount of data for each of the labels.

Test Evaluation	
Accuracy	Average Loss
93.2%	0.206643

The Test Evaluation table shows how the ASLCNN model

performed on the test data. There were actually some models that performed as well as 96% Accuracy on the test data.

6.3. LeNet-5

Once our LeNet model was implemented as described in section **5.3**, it was trained over 10 epochs in batches of 100. Using this methodology, our model is able to achieve over 99% accuracy on both dev and test splits. Below are the relevant metrics for both splits:

Dev Accuracy

Epoch	Accuracy	Average Loss
2	95.4%	0.1620
4	98.5%	0.0589
6	99.7%	0.0188
8	99.9%	0.0069
10	99.9%	0.0059

Test Accuracy

Accuracy	Average Loss
99.6%	0.0198

The model is lightweight, each epoch taking roughly 10 seconds to complete. Future results can be replicated by seeding our model.

NOTE: when saving and reloading this model, it performs significantly worse and achieves roughly 63% accuracy on the test split. Further experimentation with pytorch is required to fix this issue.

6.4. Compare and Contrast Designs

After training and analyzing the 3 models we found that SDCNN and ASLCNN have many similar characteristics. Both models contain max pooling and dropout layers to help learn the features of the ASL images. The key ways in which these designs differ are in the kernel sizing and the number of CNN layers. The effect that the kernel size would have is increasing the total number of parameters, doing this could cause overfitting if not careful. As the accuracy was not that high for SDCNN, we have tried all combinations for different kernel sizes and different CNN layers and found the best parameters for the model. Increasing the kernel size of the ASLCNN would not have helped because the model has already been very close to overfitting the training data.

The best design is the LeNet-5 it uses a kernel size of 4 which is the same as SDCNN but different then the ASLCNN.The Reason that the LeNet-5 has the ability to perform more efficiently then the SDCNN and ASLCNN is that its dropout layer uses a rate of 0.2, the same as the SDCNN causing a much lower chance of a hidden input. Aswell as having the same number of CNN layers as the ASLCNN. This allows for the model to have the best of both worlds a high number of layers and a low dropout rate giving the model the ability to learn some of the features of the dataset better then the other models.

7. Conclusion

Regardless of our specific implementation of CNNs, we were able to consistently achieve over 90% accuracy over our data. While this may be a positive indicator for applying CNNs to sign language recognition, there was still a noticeable discrepancy between the results of our models that would affect its real world applications. The difference between our implementations achieving low 90% and high 90% accuracies possibly has to do with the number of layers used. SDCNN utilized 2 convolutional layers and 2 pooling layers while achieving accuracies of roughly 92% , while the LeNet-5 implementation includes an additional convolutional layer and achieved near-100% accuracies. By introducing an additional convolutional layer with the right architecture to our SDCNN model, we could enhance its ability to articulate input images and improve its accuracy. In addition, the only model to use a dropout rate of 0.5/0.8 was our ASLCNN model as opposed to our other two models which both utilized a dropout rate of 0.2. Using this enhanced dropout rate, our ASLCNN model was able to achieve near-100% accuracies on its training data. What makes this peculiar is how similar the overall architecture of each model is: all utilize alternating convolutional layers and pooling layers, yet despite these similarities one model achieved desirable results with a massively different dropout rate.

8. Limitations

By far the biggest limitation of our models are their abilities to only read in 28 x 28 images. Real-world images of ASL gestures often have a much higher resolution, and if these images need to be passed into our model their resolution must be reduced considerably. Its unclear how representative these scaled-down images are of their original image, and factors such as how cluttered the background of these images are could limit our model's ability to tell the hand from the foreground in these reduced-resolution images. In addition, our datasets don't include the ASL gestures for "J" and "Z", since our datasets only deal with static images and both those gestures require movement. Additional challenges may arise with implementing the capability to recognize these gestures. If "J" and "Z" were to be represented using a single static image of the most distinct moment of their gesture, our model must be able to capture that single frame of those gestures correctly. This could introduce all sorts of complications in a model trained to only recognize static images. However, if "J" and "Z" were to

be represented using multiple static images of different moments in their gesture, said static images can easily be confused with other hand gestures that only have one possible configuration.

9. Future Work

This research work holds interesting prospects for further work. At first in the short term, we can design some model which can work with the series of images or else video to detect ”J” and ”Z” properly. A common method for implementing “J” and “Z” is to take a static image of the hand at the beginning of both gestures. This is fairly effective since both “J” and “Z” are distinct enough that these single frames can set them apart from other letters in the sign language alphabet.

Furthermore in the long-term, the research can be done is to use the CNN model to detect signs from the video as real-time sign language detection. So, the feed can be given as a video and predictions can be made continuously. In addition to it, the model can be trained on bigger kernel sizes, and also we can analyze the model by developing it where different kernel sizes can be used for different layers.

References

[1] Suharjito and Gunawan et al. 2018 *Sign Language Recognition Using Modified Convolutional Neural Network Model 2018 Indonesian Association for Pattern Recognition International Conference (INAPR)*. pg. 1-5

[2] K. Bantupalli and Y. Xie 2018 *American Sign Language Recognition using Deep Learning and Computer Vision 2018 IEEE International Conference on Big Data (Big Data)*. pg. 4896-4899

[3] Pigou L., Dieleman S., Kindermans P. and Schrauwen B 2014 *Sign Language Recognition Using Convolutional Neural Networks Workshop at the European Conference on Computer Vision*. pg. 572- 578

[4] Yann L., Leon B., Yoshua B., and Patrick H. 1998 *Gradient Based Learning Applied to Document Recognition Proceedings of the Institue of Electrical and Electronic Engineers*. Section II B., pg. 7-8

[5] Mohammed K., Ahmed A., and Hadeer M. 2020 *Classification of Garments from Fashion MNIST Dataset Using CNN LeNet-5 Architecture 2020 Internation Conference on Innovative Trends in Communication and Computer Engineering*. pg. 238-239