Advancing ASL Recognition with Deep Learning: A Step Towards Inclusive Communication

Rafaela Octaviani de Assunção, Kacper Szkoda

Abstract

This paper aims to compare the performance of a fully connected dense network and a convolutional neural network in a multiclass image classification task. The performance of the two models is evaluated on two custom test datasets, comparing their accuracy on new unseen data and showing each model's ability to generalize the knowledge from training datasets. Even with a much smaller number of parameters, the CNN model made much fewer mistakes on signs with similar shapes and performed better on the test data, showing its greater spatial understanding and ability to generalize learned information to new data.

1 Introduction

Hearing disabilities affect 5% of the world's population, yet the language barrier between them and the hearing population significantly affects their quality of life. Sign language, the main way people with hearing disabilities communicate, is hardly taught to the hearing population in schools, which makes communication hard between these groups. Therefore, we hope to help bridge that gap with deep learning by translating the ASL alphabet from images to text.

To achieve this, we aim to design a machine learning model capable of recognizing ASL letters based on image data. This project aims to compare the performance of a fully connected dense network and a convolutional neural network in a multiclass classification task. Both models take images of ASL hand signs as their input and output the corresponding letter/symbol of the sign. The input images are resized and gray scaled to lower the number of input nodes, avoiding excessive computation cost, and normalized for better numerical stability.

This paper aims to compare the performance of the two proposed models and show that a convolutional model performs significantly better than a fully connected one in image classification. We prove this by benchmarking the two models on some test datasets and comparing their accuracy on new unseen data, also showing each model's ability to generalize the knowledge from the training datasets.

2 Related Work

Plenty of techniques have been used to approach the image classification of ASL hand signs. Barbhuiya, A.A., Karsh, R.K., Dutta, S. [1] used an AlexNet CNN to classify ASL sign images. This type of architecture is much deeper than the CNN used in this paper, having more convolutional layers, cross-channel normalization layers before max-pooling layers, and a much larger linear network at the end. The paper also uses a dataset without backgrounds, extracting only the hand, which would eliminate the effect of the background when testing the model in unseen data. Sharma, S., Kumar, K. [2], use a 3-D CNN to classify entire videos of dynamic signs, splitting the video into key frames which are all fed into the model. Using videos also opens up the possibility of fully translating ASL instead of just the alphabet, as some more complicated signs involve movement. Ma, Y., Xu, T., Kim, K. [3] use a two-stream mixed CNN, which takes two images as input. This allows the model to use two consecutive images for dynamic signs while taking two of the same image for static signs, also opening up the number of signs it can recognize.

Jiang, D., Li, G., Sun, Y. et al [4] focus more on the image preprocessing before feeding it to the CNN. They use a binary graph turning the image into only 0 values for the background and 1s for the hand, combined with a refinement algorithm that removes layers of pixels until a thin skeleton of the hand is left, still showing the position of all the fingers and the wrist. This approach makes calculations lighter for the model as it only needs to calculate the positions of 1s.

Barbhuiya, A.A., Karsh, R.K., Jain, R. [5] take a different approach and use a pretrained CNN for feature extraction, then a support vector machine (SVM) for classification. This approach skips the work of training the model, using a state-of-the-art VGG16 model to turn the input images into a feature map.

3 Dataset and Features

The dataset used was found on Kaggle [6]. It consists of 29 classes, representing all letters from the English alphabet, extended to include signs for delete, space, and nothing - no sign, empty space. There are 87,000 images, 3000 per class. An 80 - 20 split was chosen for training and validation data. The test data comes from a custom dataset of photos of our hands. All samples are 200x200 pixel RGB images, which already presents an implementation challenge. Each image is originally represented by 40000 pixels per color channel, totaling 120000 data points per image. To keep the number of trainable parameters to a reasonable level and decrease the computation time necessary for training the models, the image preprocessing pipeline is composed of resizing, grayscaling and normalizing, which significantly decreases the number of inputs for the models and rescales the pixel values from the 0-255 range into 0-1. Fine details of the images are not important for the classification, as hand signs differ by significant shape changes, which also explains why grayscaling does not present important information loss. In the figure below, a sample of the training dataset is presented, images together with their respective labels:

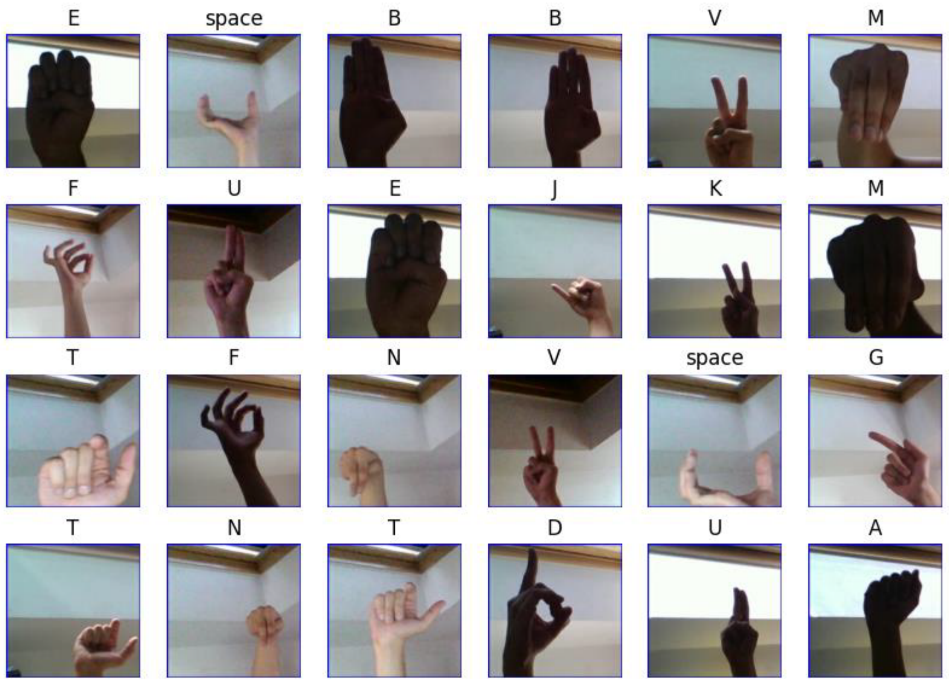

Figure 1: Dataset showcase.

After examining the images, an important characteristic of the dataset can be deduced. All images are of the same hand in a set of identical backgrounds. The only features that differ are the lighting and position of the hand. This homogeneity of the dataset can hinder the models' ability to generalize to new distributions.

To test this hypothesis, we created two small testing datasets containing around 150 images each, evenly distributed among the classes. One is set in a white consistent background, and the second is designed to be more challenging, with a diverse background containing multiple shapes and textures.

4 Methods

Two different architectures are trained for the task and compared. The first is a fully connected deep neural network and the second is a convolutional neural network [7].

A fully connected network is a subtype of a neural network, specifically a computational model inspired by both the function and structure of the brain, from which it derives its name. It consists of neurons organized into layers, responsible for processing inputs and passing results further. Each input is multiplied by a learned weight, all are summed, and a learned bias is applied. This term is inputted into an activation function in order to normalize and introduce non-linearity, and the output is fed into further neurons. Through this, certain neurons respond to specific features. A fully connected network in particular is characterized by every neuron in one layer being connected to every neuron in the next layer, allowing it to represent more complex relationships:

$$a^{(l)} = \sigma(W^{(l)}a^{(l-1)} + b^{(l)}) \tag{1}$$

Where $a^{(l)}$ is the l-th layer of nodes, $W^{(l)}$ and $b^{(l)}$ are the weights and biases of layer l, and $\sigma()$ is the activation function (ReLU in this case). A convolutional neural network is a design particularly fit to process data in grid-like structures. This is achieved through CNNs utilizing kernels, which are matrices with learnable parameters, most often small, typically 4x4 or 3x3 values. The convolution in the name stems from the kernels sliding across input data, essentially performing a convolution operation, mathematically described by:

$$(I * K)(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(i-m, j-n) \cdot K(m,n) \tag{2}$$

Where $(I*K)(i,j)$ is the convolution of image I and kernel K at the point (i,j), and M and N are the dimensions of the kernel. This can be conceptually understood as the kernel, for each pixel, also considering the surrounding pixels, and learning the relationship between them. Due to this, the kernels often turn to represent structures; simpler ones, like edges or shapes, but also more complex ones in the deeper layers, such as textures or even uninterpretable relationships only the machine can see.

Each convolutional layer can contain multiple kernels, each of which produces a feature map; these are stacked together to form a tensor as input for the next convolutional layer. After a convolutional layer, a max-pooling layer is used, to reduce the spatial dimensionality of the data. This allows for reducing complexity while retaining the most important information. The next convolution then operates on the entire tensor, each kernel on all the feature maps, which could mean, for example, combining different detected edges into detecting an entire shape.

There are several benefits to using CNN over FC for image processing. While FC networks treat each pixel separately, CNNs focus heavily on spatial relationships, only having weights for the kernels. This is called weight sharing, and can drastically reduce the number of parameters a network includes, especially at higher resolutions. In our case, it was the difference between 880,029 parameters for 40x40 pixel FC and 41,117 for CNN 50x50 pixel. Another benefit is how focusing on relationships makes it easier to generalize, this includes being more robust to translations, color changes, different backgrounds. Additionally, the previously mentioned feature combinations allow for deeper understanding in later layers, necessary for complex datasets.

The specific final model architectures can be seen below:

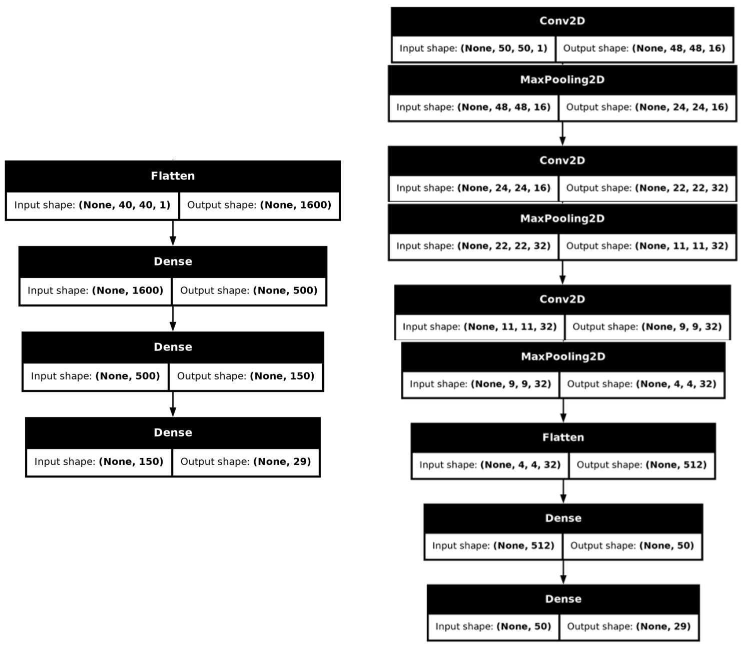
Flatten
Input shape: (None, 40, 40, 1)
Output shape: (None, 1600)
Dense
Input shape: (None, 1600)
Output shape: (None, 500)
Dense
Input shape: (None, 500)
Output shape: (None, 150)
Dense
Input shape: (None, 150)
Output shape: (None, 29)
Conv2D
Input shape: (None, 50, 50, 1)
Output shape: (None, 48, 48, 16)
MaxPooling2D
Input shape: (None, 48, 48, 16)
Output shape: (None, 24, 24, 16)
Conv2D
Input shape: (None, 24, 24, 16)
Output shape: (None, 22, 22, 32)
MaxPooling2D
Input shape: (None, 22, 22, 32)
Output shape: (None, 11, 11, 32)
Conv2D
Input shape: (None, 11, 11, 32)
Output shape: (None, 9, 9, 32)
MaxPooling2D
Input shape: (None, 9, 9, 32)
Output shape: (None, 4, 4, 32)
Flatten
Input shape: (None, 4, 4, 32)
Output shape: (None, 512)
Dense
Input shape: (None, 512)
Output shape: (None, 50)
Dense
Input shape: (None, 50)
Output shape: (None, 29)

Figure 2: Comparison of FC model architecture and CNN model architecture.

5 Results and Discussion

Both models and the process of their training will be discussed. Accuracy is the most important metric examined. As the task is multi-class classification aiming to compare two architectures differing in their spatial understanding of data, confusion matrices will be used to interpret the results.

The first model trained was the FC model, the specific architecture for which was obtained from Assignment 4, multiclass classification using neural networks. The changes made include introducing a layer resizing into 40x40 pixel and a rescaling layer for pixel values (the dataset used in that Assignment is rescaled already). The numbers of neurons for the dense layers were also adjusted to better fit the current input size.

The hyperparameters used for the model training were: batch size: 128, learning rate: 10^{-4}. These were chosen after previous runs showed oscillations, suggesting overshooting. Larger batch sizes allow the updates to be based on a more averaged out loss, while slower learning rates help prevent a local minimum from being overshot by decreasing step size. The obtained loss, in log and linear scales, and accuracy plots are presented below:

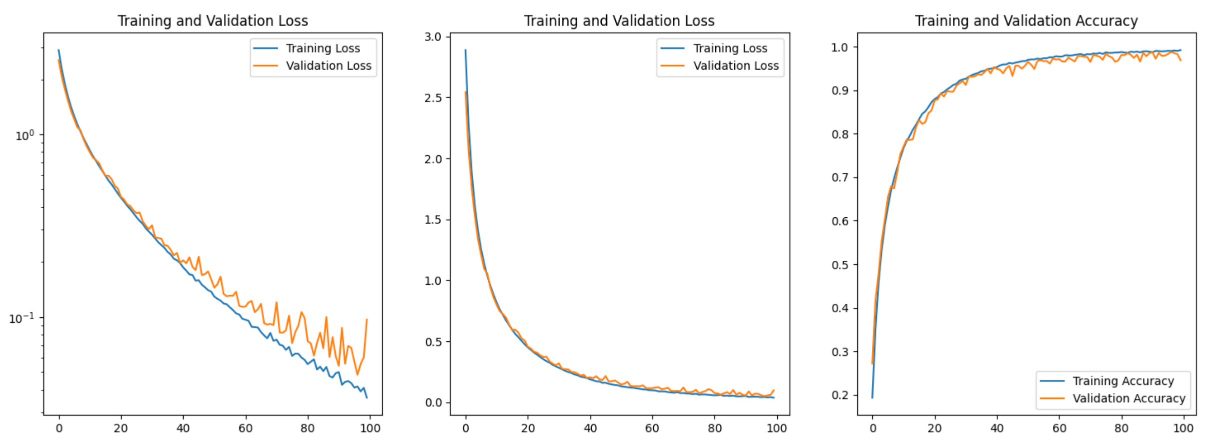

Figure 3: Loss (log (left) and linear (middle) scales) and accuracy plots for FC network training.

It can be seen that convergence is obtained for both the training and validation loss. There are oscillations and a small, yet increasing, gap between the training and validation losses going towards convergence. The final accuracy in the validation data is 96.9%. To examine the misclassified images, a confusion matrix was created:

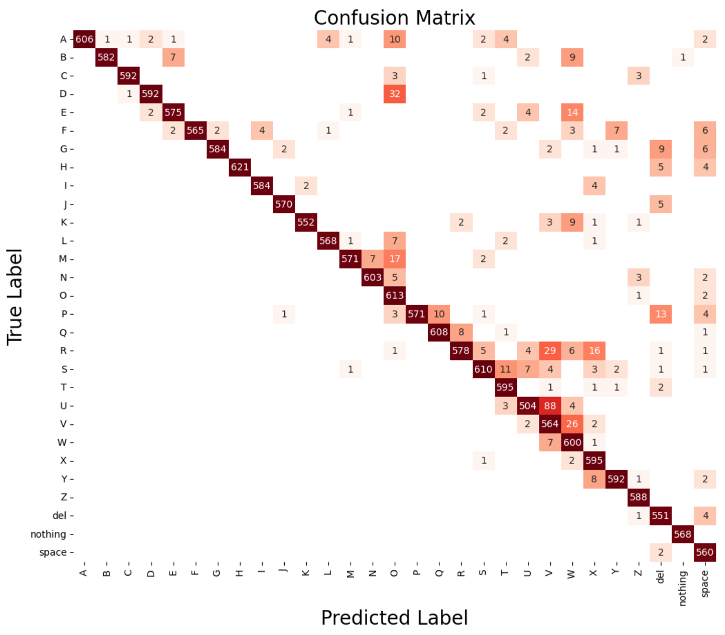

Figure 4: FC confusion matrix.

Examining the most prominent mistakes, a clear pattern emerges:

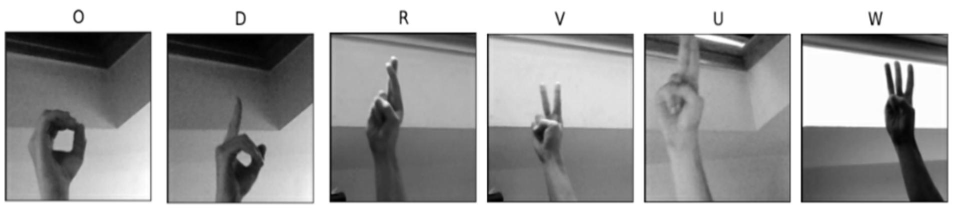

Figure 5: Showcase of most confused classes.

The FC model seems to struggle with telling apart ASL signs involving similar shapes, such as ones differing just by the number of fingers being held up.

To train the CNN, we conducted a model selection step to choose an architecture that minimizes validation loss.

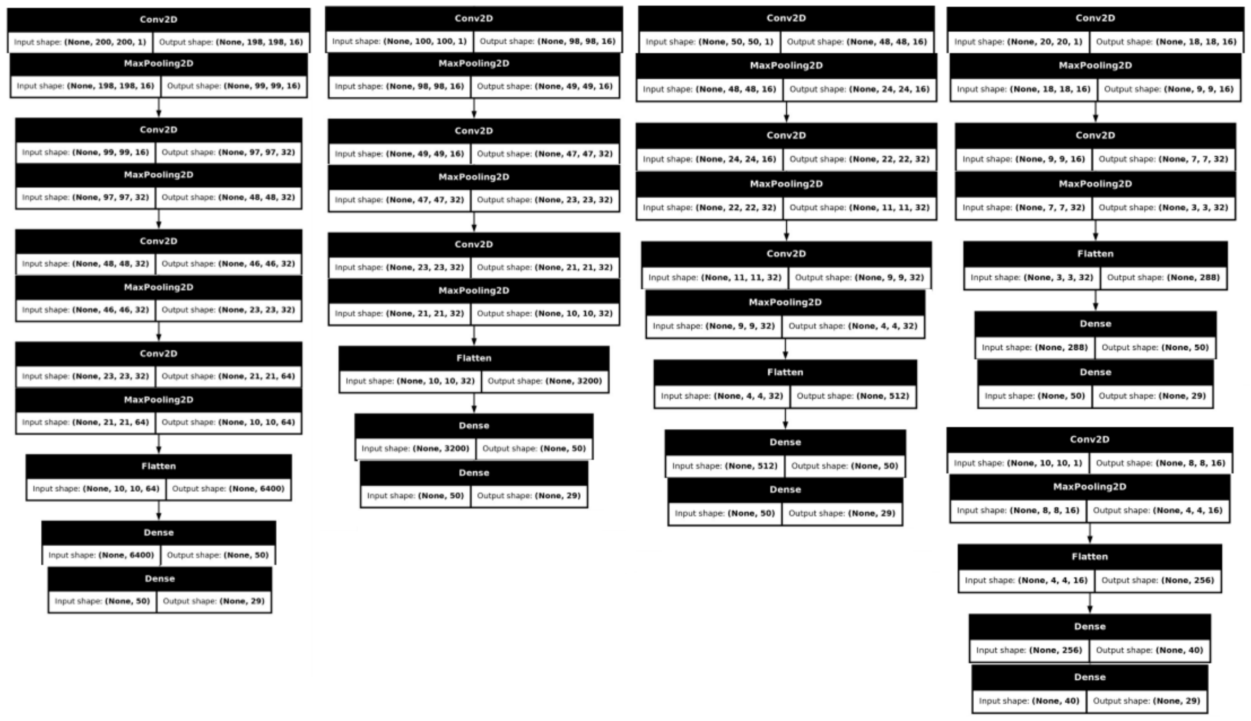
Conv2D
Input shape: (None, 200, 200, 1) Output shape: (None, 198, 198, 16)
MaxPooling2D
Input shape: (None, 198, 198, 16) Output shape: (None, 99, 99, 16)
Conv2D
Input shape: (None, 99, 99, 16) Output shape: (None, 97, 97, 32)
MaxPooling2D
Input shape: (None, 97, 97, 32) Output shape: (None, 48, 48, 32)
Conv2D
Input shape: (None, 48, 48, 32) Output shape: (None, 46, 46, 32)
MaxPooling2D
Input shape: (None, 46, 46, 32) Output shape: (None, 23, 23, 32)
Conv2D
Input shape: (None, 23, 23, 32) Output shape: (None, 21, 21, 64)
MaxPooling2D
Input shape: (None, 21, 21, 64) Output shape: (None, 10, 10, 64)
Flatten
Input shape: (None, 10, 10, 64) Output shape: (None, 6400)
Dense
Input shape: (None, 6400) Output shape: (None, 50)
Dense
Input shape: (None, 50) Output shape: (None, 29)
Conv2D
Input shape: (None, 100, 100, 1) Output shape: (None, 98, 98, 16)
MaxPooling2D
Input shape: (None, 98, 98, 16) Output shape: (None, 49, 49, 16)
Conv2D
Input shape: (None, 49, 49, 16) Output shape: (None, 47, 47, 32)
MaxPooling2D
Input shape: (None, 47, 47, 32) Output shape: (None, 23, 23, 32)
Conv2D
Input shape: (None, 23, 23, 32) Output shape: (None, 21, 21, 32)
MaxPooling2D
Input shape: (None, 21, 21, 32) Output shape: (None, 10, 10, 32)
Flatten
Input shape: (None, 10, 10, 32) Output shape: (None, 3200)
Dense
Input shape: (None, 3200) Output shape: (None, 50)
Dense
Input shape: (None, 50) Output shape: (None, 29)
Conv2D
Input shape: (None, 50, 50, 1) Output shape: (None, 48, 48, 16)
MaxPooling2D
Input shape: (None, 48, 48, 16) Output shape: (None, 24, 24, 16)
Conv2D
Input shape: (None, 24, 24, 16) Output shape: (None, 22, 22, 32)
MaxPooling2D
Input shape: (None, 22, 22, 32) Output shape: (None, 11, 11, 32)
Conv2D
Input shape: (None, 11, 11, 32) Output shape: (None, 9, 9, 32)
MaxPooling2D
Input shape: (None, 9, 9, 32) Output shape: (None, 4, 4, 32)
Flatten
Input shape: (None, 4, 4, 32) Output shape: (None, 512)
Dense
Input shape: (None, 512) Output shape: (None, 50)
Dense
Input shape: (None, 50) Output shape: (None, 29)
Conv2D
Input shape: (None, 20, 20, 1) Output shape: (None, 18, 18, 16)
MaxPooling2D
Input shape: (None, 18, 18, 16) Output shape: (None, 9, 9, 16)
Conv2D
Input shape: (None, 9, 9, 16) Output shape: (None, 7, 7, 32)
MaxPooling2D
Input shape: (None, 7, 7, 32) Output shape: (None, 3, 3, 32)
Flatten
Input shape: (None, 3, 3, 32) Output shape: (None, 288)
Dense
Input shape: (None, 288) Output shape: (None, 50)
Dense
Input shape: (None, 50) Output shape: (None, 29)
Conv2D
Input shape: (None, 10, 10, 1) Output shape: (None, 8, 8, 16)
MaxPooling2D
Input shape: (None, 8, 8, 16) Output shape: (None, 4, 4, 16)
Flatten
Input shape: (None, 4, 4, 16) Output shape: (None, 256)
Dense
Input shape: (None, 256) Output shape: (None, 40)
Dense
Input shape: (None, 40) Output shape: (None, 29)

Figure 6: Five candidate model architecture (model 0 to 4 (bottom right)) with different input sizes and number of parameters.

The loss and accuracy plots when training each model are:

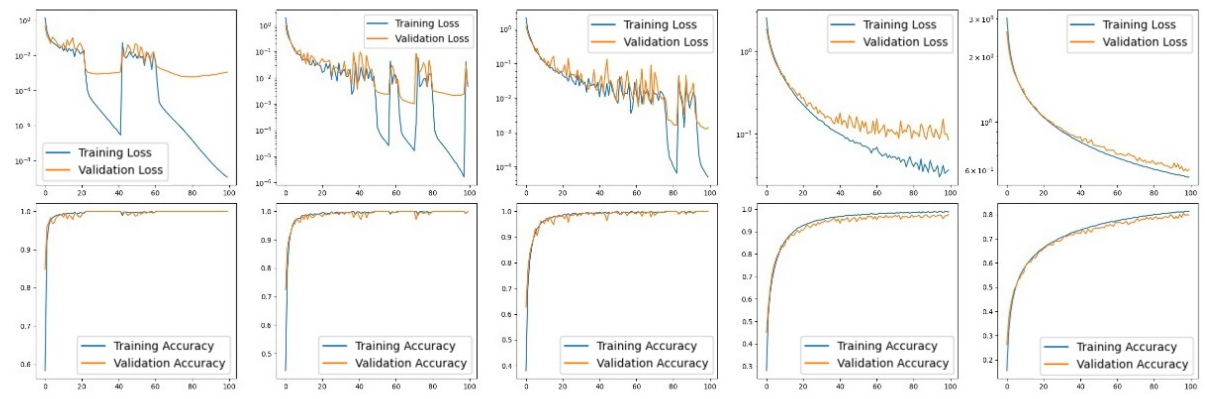

Figure 7: Loss (top) and accuracy (bottom) training plots for the model selection step (model 0 to 4 left to right).

The smallest model (model 4) showed a low accuracy, indicating the input image size of 10x10 was too small to make accurate predictions. Model 3 had better accuracy but still overfit the training data, as shown by the validation loss being higher than the training loss even at low epochs, indicating the image size is still too small for the model to learn enough information to generalize. Thus, we chose to keep model 2, as it is the smallest model with an input size large enough that the model does not overfit too early into training.

Then, we went through a grid search for hyperparameter tuning to decide between batch sizes of 128 and 256 and learning rates 10^{-4} and 10^{-5}. The loss plots are compared to select the best:

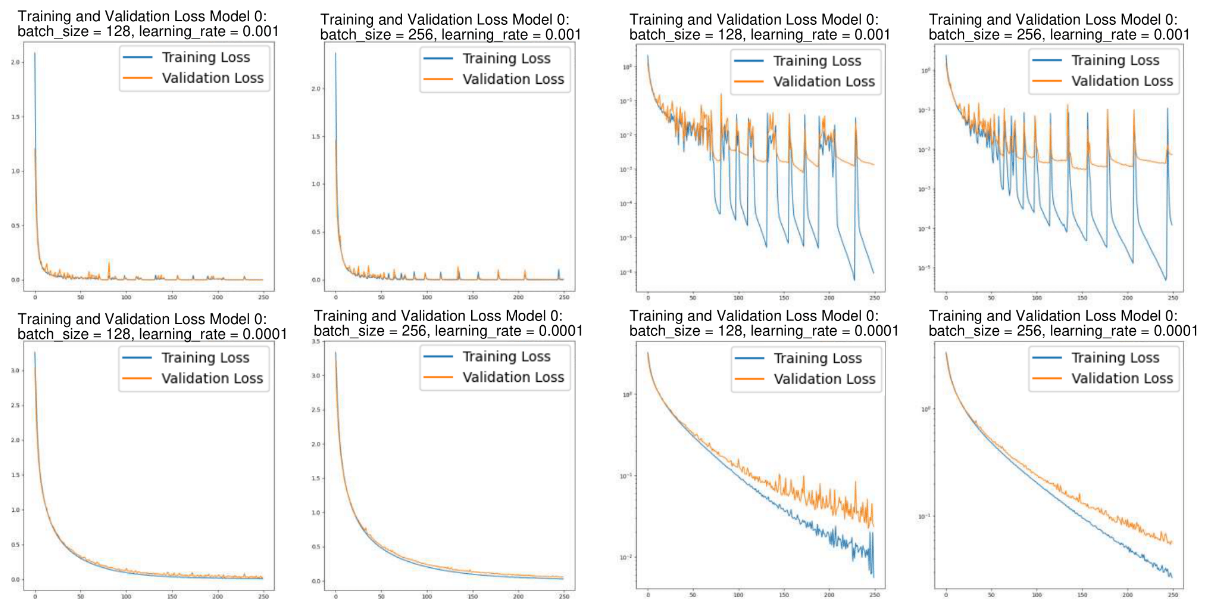

Figure 8: Linear (left) and log (right) scale loss plots from the hyperparameter tuning step.

In the loss plots it is important to notice the large oscillations, these suggest overshooting, which can stem from too large learning rates, and can also be improved by averaging out the images by utilizing larger batch sizes. For these reasons, 256 was selected for the batch size and 10^{-5} for the learning rate.

The model was re-trained for 300 epochs with the following results:

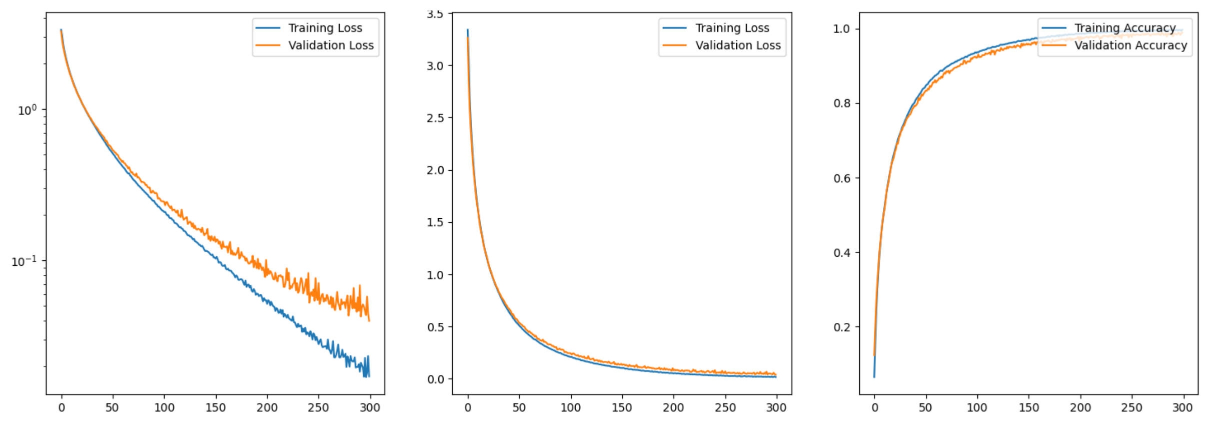

Figure 9: Chosen CNN loss (log (left) and linear (middle) scales) and accuracy plots.

The results seem almost indistinguishable from the FC model. The final accuracy in the validation data is 98.9%. The confusion matrix was examined to look into the differences further:

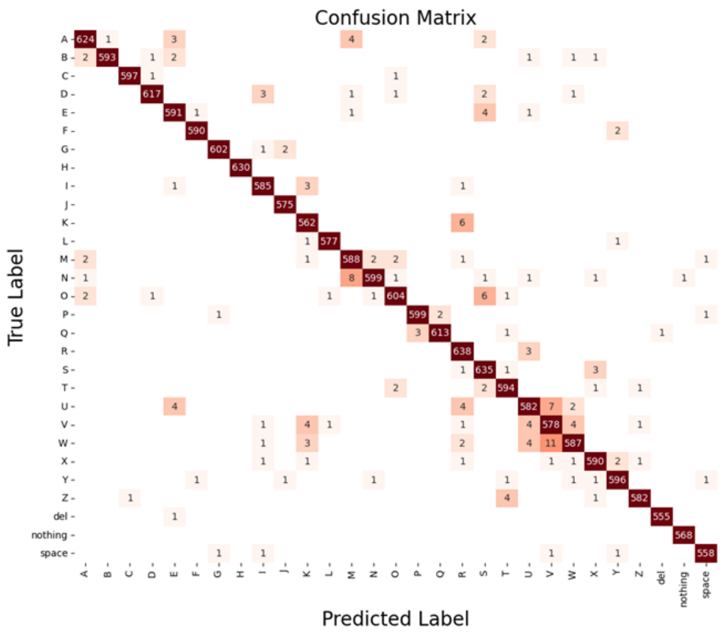

Figure 10: FC confusion matrix

Most of the previous confusion is completely gone, with a much better spatial understanding of the data.

Finally, both models were tested against the two new distributions. Example images are shown below:

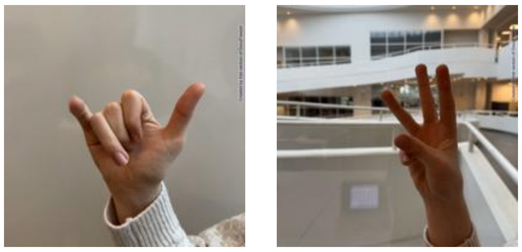

Figure 11: Y in background one and W in background two

The results are presented in the table below:

Table 1: FC and CNN accuracy across different backgrounds.

Background Type	**FC Accuracy (%)**	**CNN Accuracy (%)**
Uniform Background	5.1	22.6
Diverse Background	5.6	10.6

These results prove the networks have a very hard time generalizing to new distributions, especially with added complexity. The CNN does considerably better at both, which suggests it does understand the shapes defining the class, as opposed to the FC network, which performs only a little better than guessing at both.

6 Conclusion and Future Work

From the obtained results, it can be seen why CNN architectures are chosen for most image classification tasks. Not only do they perform better, but they are also more memory efficient. Their focus on relative positions rather than absolute values proves crucial in generalizing to new distributions. However, the CNN still did not generalize enough to perform well in the custom datasets

A reason for the poor performance on new distributions could be the homogeneous structure of the dataset. The validation loss plots converging later and at higher loss than training could suggest overfitting. Even though the dataset is large, it is very limited in terms of variability; a possible improvement could have been utilizing data augmentation techniques, such as noise injection, to increase diversity. Another possibility might have been weight regularization, which discourages large weights, typically present in overfitted, overly complex models.

A mistake we made was using the same validation data both during training and to assess the performance of the model at the end. A third distinct test set should have been defined to ensure that the performance is tested on never-before-seen data. While the validation dataset is not used for weight updates, which means it can still provide an overfitting benchmark, its accuracy is being constantly tracked.

Future work could include utilizing the CNN model in transfer learning to improve its generalizing potential by training the higher layers on a more diverse dataset. Once the performance is satisfactory, such models can be used in a pipeline selecting video frames to translate ASL letters in near real-time. Another possibility could be to translate words and phrases instead; in that case, a much more extensive dataset and more sophisticated architecture that supports dynamic signs, like the 3-D CNN [2] and the two-stream mixed CNN [3] mentioned in the related works section, would be needed.

Contributions

The entirety of the project was done in close collaboration, due to all the tasks being tied together. All decisions were discussed extensively and taken after mutual agreement. In terms of labor division, most of the code was written and run by Rafaela, while the bulk of the paper was written by Kacper. However, both of us contributed to every step.

References

[1] A. A. Barbhuiya, R. K. Karsh, and S. Dutta, “Alexnet-cnn based feature extraction and classification of multiclass asl hand gestures,” in *Proceeding of Fifth International Conference on Microelectronics, Computing and Communication Systems*, V. Nath and J. K. Mandal, Eds., Singapore: Springer Singapore, 2021, pp. 77–89, ISBN: 978-981-16-0275-7.

[2] S. Sharma and K. Kumar, “Asl-3dcnn: American sign language recognition technique using 3-d convolutional neural networks,” *Multimedia Tools and Applications*, vol. 80, no. 17, pp. 26 319–26 331, 2021.

[3] Y. Ma, T. Xu, and K. Kim, “Two-stream mixed convolutional neural network for american sign language recognition,” *Sensors*, vol. 22, no. 16, p. 5959, 2022.

[4] D. Jiang, G. Li, Y. Sun, J. Kong, and B. Tao, “Gesture recognition based on skeletonization algorithm and cnn with asl database,” *Multimedia Tools and Applications*, vol. 78, no. 21, pp. 29 953–29 970, 2019.

[5] A. A. Barbhuiya, R. K. Karsh, and R. Jain, “Cnn based feature extraction and classification for sign language,” *Multimedia Tools and Applications*, vol. 80, no. 2, pp. 3051–3069, 2021.

[6] *Asl alphabet.* DOI: 10.34740/KAGGLE/DSV/29550. [Online]. Available: https://www.kaggle.com/dsv/29550.

[7] J. Wu, “Introduction to convolutional neural networks,” *National Key Lab for Novel Software Technology. Nanjing University. China*, vol. 5, no. 23, p. 495, 2017.