Classifying ASL Letters with Neural Networks

Jake Browning, with guidance from Torey Hilbert

Department of Mathematics, The Ohio State University

Abstract

According to some estimates, there are between 100,000 to 1,000,000 speakers of American Sign Language (ASL) in the United States. Quickly translating ASL to English is a difficult skill that can take hundreds or thousands of hours to learn and refine. So, developing an automated method of recognizing signs using neural networks from scratch would almost certainly involve a significant investment of time and resources. This project aims to provide a comparison of some approaches to a simpler version of the task. A dense neural network, and a convolutional neural network (CNN) were designed and trained to identify the letters of the ASL alphabet using the tensorflow machine learning library, numpy, and the Sign Language MNIST data set. The models' performances were compared based on accuracy and loss metrics. It was observed that the CNN outperformed the dense network in classifying most letters, despite having fewer parameters.

Neural Networks

A neural network consists of **layers** of **nodes**. Each node in a layer of nodes takes the outputs of the nodes in the previous layer as inputs. Similarly, their outputs act as the inputs for the next layer. This is how a network takes **features** and turns them into **labels**. However, the output of the network is not useful without training. **Training** is the process by which the weights of a network are calibrated so that it can produce useful results.

Definitions

A **neural network**, $f : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$, is defined as follows:

1. Let $x_0 \in \mathbb{R}^{d_{in}}$ be the input to the network.
2. Let $z_{k+1} = W_k x_k + b_k$ where $k = 0, \ldots, n$.
3. Let $x_{k+1} = \sigma(z_{k+1})$.
4. Let $f(x_0)_i = \frac{e^{z^i_{n+1}}}{\sum_{j=1}^{d_{out}} e^{z^j_{n+1}}}$

Where x_k is the **output** of the **layer k** and the **input** of **layer k+1**, z_k is the pre-activation output of layer k, σ is an **activation function**, the matrix W_k is a **weight**, b_k is a **bias**, and f_i is a **layer**.

During training, the network is adjusted according to another function, called an **optimizer function**. This process is called **gradient descent**.

Convolutional Neural Networks (CNNs)

A **convolutional neural network** or **CNN** is a type of neural network that makes use of convolutional layers. A convolutional layer considers only certain inputs from the previous layer. This is different from a dense layer, which considers every input from the previous layer. A convolutional layer considers inputs that are near to each other in groups. For example, instead of the taking all the values of every pixel in an image as an input, a convolution may take the maximum value of each 3x3 grid of pixels. This reduces the computational load of feature extraction. As such, convolutional layers are typically used to rapidly extract low-level features from a data set, such as the lines or shapes in an image.

Convolutions are particularly useful in image processing, as convolutional networks are **translationally equivariant**. This means that an input can be slightly shifted before or after being sent through a convolution and the end result will be the same. This means that CNNs do not overreact to small shifts in the appearance in an image. Because of this equivariance property, we would expect a CNN to outperform a dense network of a similar size and complexity in an image classification task.

Method

There were two networks tested in this experiment. The first was a nine-layer convolutional neural network, with three convolution layers. The second was a four-layer dense neural network. The former had 24,742 parameters, and the latter had 101,326 parameters. These networks were compiled and training using tensorflow and keras, while the data were processed using numpy. All coding was done in python.

Both models were trained on the **Sign Language MNIST** data set, which has around 35,000 images of ASL letters, excluding J and Z, at a resolution of 28 by 28 pixel. These networks were optimized using sparse categorical crossentropy as the loss function and adam as the optimizer. Both networks were trained on the same machine for similar amounts of time and both used a batch size of 50. The convolutional neural network was trained for 150 epochs. The total training time was 32 minutes, or 7 hours and 41 minutes in CPU time. The dense neural network was trained for 1261 epochs. The total training time was 35 minutes, or 7 hours and 53 minutes in CPU time.

Results

When evaluated using the test data, the convolutional neural network had an accuracy of 89.24% and a loss of 2.0904, and the dense neural network had an accuracy of 7.63% and a loss of 4.9333. Both networks performed better than random guessing, which has an accuracy of 4.16%. The convolutional network reached its minimal loss in epoch 105, with a loss of 3.8209e-10, and an accuracy of 100.00%. The dense network reached its minimal loss in epoch 76, with a loss of 3.1063e-05, and an accuracy of 100.00%.

In the last third of the training session, the dense network rapidly declined in performance. It is suspected that this was a result of an inappropriately large training step size. Such an error could cause the model to repeatedly overshoot the optimal network state, resulting in the type of oscillation visible in the Figure 1, below. If the error described is what occurred, we could say that the network had **exploding gradients**.

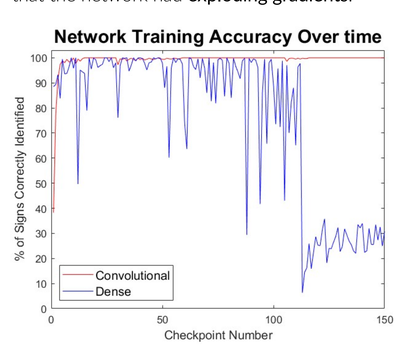

Figure 1. A comparison of the accuracy of the dense and convolutional networks as training progressed. Each checkpoint corresponds to approximately 21 seconds of training in CPU time.

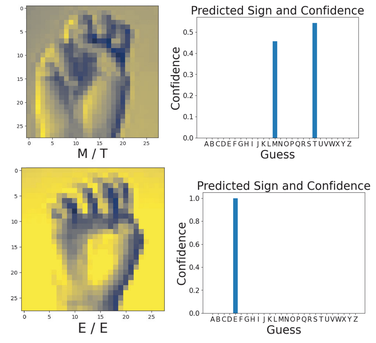

Figure 2. Two sets of features are displayed as images (left). The text below each graphs shows the correct answer and the CNN's guess in the format "Answer/Guess". The bar graph shows the model's confidence that a given letter is the correct label for the feature (right).

Despite this error, the dense network was still somewhat capable of identifying the letters M, R, C, and S. The dense network had an accuracy of 71.32%, 63.89%, 30.32%, and 19.32% when given the letters M, R, C, and S, respectively.

Interestingly, the dense network was more accurate than the CNN when given the letter R, as the CNN correctly identified R only 56.25% of the time. This was the letter that the CNN had the lowest accuracy in guessing, with the second lowest being G, at 68.39%, then T at 72.98%. The CNN was over 80% accurate in classifying each of the remaining letters, and correctly classified every instance of the letters A, F, L, and P in the test data.

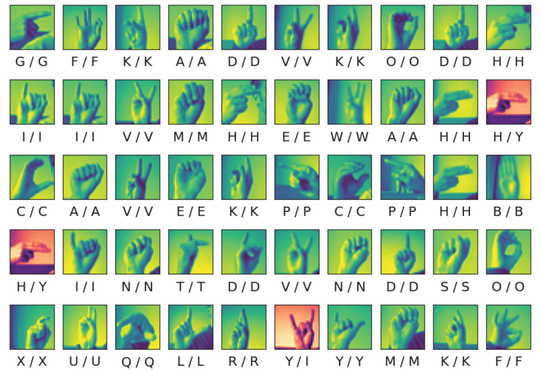

Figure 3. The first 40 sets of features in the test set, and a collection of 10 interesting sets of (which are, in order, sets 41, 52, 54, 62, 87, 88, 117, 124, 148, and 789). The text below each graphs shows the correct answer and the CNN's guess in the format "Answer/Guess". Correctly guessed signs are tinted green, while incorrectly guessed signs are tinted red.

Conclusions

Overall, the Convolutional Neural Network performed much better than the dense neural network. It more improved more rapidly in terms of CPU time during training, and was more accurate in predicting the labels in the test data.

However, with a slower rate of training, it is possible that the dense network would be able to avoided the exploding gradients that are suspected of causing its accuracy to plummet. Another factor that could unfairly favor the CNN is the length of training time. Since the dense network has more parameters, it is reasonable to expect that it would take longer for the network to be trained to a similar level as the smaller CNN.

On the other hand, the resolution of the image favors the dense network. A higher resolution image would allow the convolutions used by the CNN to be able to discard more information between layers relative to the dense network. This means that a dense network trained to identify high quality images would require more CPU time to train than a CNN with a similar number of layers.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Chollet, F. et al. (2015). Keras. `https://keras.io`.

Eberhard, David M., G. F. S. and Fennig, C. D. (2022). American sign language. `https://www.ethnologue.com/25/language/ase/`.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

MATLAB (2022). *version 9.12.0 (R2022a)*. The MathWorks Inc., Natick, Massachusetts.

pandas development team, T. (2020). pandas-dev/pandas: Pandas.

tecperson (2018). Signlanguage mnist. `https://www.kaggle.com/datasets/datamunge/sign-language-mnist`.