**Build-Your-Own CNN**

**Explain how you split the data, either by describing what you did, or by showing the code that you used. Justify your choice of splitting strategy. How many training, validation, and test images do you have?**

When splitting the data, we must ensure that there is enough training data for the model to be able to tune parameters, such as kernel weights, properly. At the same time, we must withhold enough data so that there are still images that the model has not seen. These images are used for validation purposes to find the best hyperparameters (batch size, learning rate, model architecture), and for testing to see how well the model can classify ASL gestures to their corresponding letters.

I manually split the data into a 60:20:20 ratio of training:validation:testing images. Each dataset had 9 classes for the letters A through I.

Following the ratio above:

* There are 156 training photos, 59 validation photos, and 57 testing photos for A, B, C, E
* There are 156 training photos, 60 validation photos, and 56 testing photos for D, F, G, H
* There are 147 training photos, 51 validation photos, and 51 testing photos for I

Within each class, I kept photos belonging to the same person together. This was done to ensure that the model would see different gestures made by different people when the training, validation, and testing datasets were passed as input. By preventing the model from fitting to people’s hands, it is less likely that the model will overfit. This is also why the ratios are not exactly 60:20:20, but rather 57:22:21 or 59:20:20 depending on the letter.

**Explain your choice of neural network architecture: how many layers did you choose? What types of layers did you use? Were they fully-connected or convolutional? What about other decisions like pooling layers, activation functions, number of channels / hidden units?**

This NN contains 2 convolutional layers, 2 max-pooling layers, and 2 fully-connected layers.

Samples pass through the convolutional layer, into the activation function, and through the max-pooling layer twice before entering the fully-connected layers.

The first convolutional layer has 3 in-channels and produces 5 feature maps by convolving the input with a 5x5 kernel. The max-pooling layer reduces the dimensions of the input by convolving it with a 2x2 kernel and stride of 2. The second convolutional layer has 5 in-channels and produces 10 feature maps again through convolution with a 5x5 kernel. The max-pooling layer used after the second convolutional layer is identical to the previous one. I chose to use a 5x5 kernel, but the kernel size can always be adjusted to be 3x3 instead (and this is what I did in a subsequent part of the assignment).

In between the convolutional layer and the max-pooling layer, I chose the ReLU activation function because it is better at training networks where there are many nodes. The ReLU function also prevents the vanishing gradient problem from occurring during back propagation.

I chose to use 2 fully-connected layers to process the results of the 10 output feature maps and classify them into one of the 9 classes. Because the previous convolutional layers did a lot of processing, there is not a need for many fully-connected layers - otherwise, the model is prone to overfitting. In between the fully-connected layers, there are 32 hidden units.

**Explain your choice of loss function and optimizer.**

Because this model must classify samples as 9 different letters (a multi-class classification problem), I chose to use the cross-entropy loss function. If the model’s predicted probability diverges from the actual outcome, then the cross-entropy function would produce a high loss value. The model must then minimize this loss in the backpropagation to produce a more accurate prediction in the next forward pass.

I used Adam as my optimizer. Adam is an extension of SGD that also incorporates benefits of both the AdaGrad and RMSProp algorithms. Rather than adjusting the parameter learning rate using the mean, Adam utilizes a version of the variance. The Adam optimizer is also faster than SGD.

**Choose the best model out of all the ones that you have trained. Justify your choice.**

I chose to test the model using a batch size of 128, a learning rate of 0.001, and a 5x5 kernel for convolution. These parameters combined gave me a training accuracy of 0.996 and a validation accuracy of 0.64 after 30 epochs. The training loss was 0.053 and the validation loss was 1.72.

This model had a comparable training accuracy to the baseline model (batch size = 64 and learning rate = 0.001) and the validation accuracy was only slightly less. However, the validation loss of this model did not diverge as much as the baseline model once the number of epochs increased.

Nevertheless, when compared to models that use a 3x3 convolutional kernel, there is still a large divergence between the training loss and the validation loss after 10 epochs, suggesting that the model is prone to overfitting under these parameters. However, the models that used a 3x3 convolutional kernel had a training accuracy of around 0.7, which is lower than the training accuracy this model produced.

All together, this suggests that it is difficult to balance overfitting with good accuracy in CNNs.

**Transfer Learning**

**Explain your choice of neural network architecture: how many layers did you choose? What types of layers did you use: fully-connected or convolutional? What about other decisions like pooling layers, activation functions, number of channels / hidden units in each layer?**

The idea behind transfer learning is to use an existing CNN that was pre-trained on a large dataset (like how AlexNet was trained on more than 1 million images from the ImageNet database) to extract features for the task to be performed.

As such, I did not use any further convolutional layers, and I only used 1 fully-connected layer to fine-tune the weights that were pre-trained by AlexNet in one final backpropagation step. In my fully-connected layer, I did not use any pooling layers.

Similar to the first half of the assignment, I used the ReLU activation function to prevent the vanishing gradient problem from occurring during the backpropagation phase. My fully-connected layer contains 32 hidden units.