# Implementation

1. Dataset

The dataset consisted of a collection of hand-written character images, where each image contained either a letter in the range of A-H, the dollar sign symbol ($), or the hash symbol (#). In order to ensure a richer dataset and consequently a more generalizable model, the handwritten characters were purposely collected to account for variability. This includes the writing utensil used, whether the letter was lower or upper case, and whether the letter was written in print or cursive. In total, the labelled training dataset was comprised of 6720 300x300 images. The raw images were resized to 300x300 and set to grayscale in order to create the labelled training dataset.

1. Preprocessing

A necessary condition for achieving an accurate CNN model is image preprocessing. Image preprocessing was accomplished through the use of the OpenCV library, which provides several morphological operations that can be performed on image data. Morphological operations require the use of a kernel, which describes the neighborhood of pixels that are considered around each pixel when performing a given operation. It was found that for the given dataset, a 4x4 kernel matrix performed best with the given operations.

Our implementation utilized two morphological operations. Median blur was the first operation applied on the data, which takes the average of the neighboring pixels to calculate a new pixel value for the output. This had the effect of smoothing the edges of each character. Next, the MORPH\_OPEN operation was performed, which is a combination of erosion, an operation which qualitatively “thins” the characters, and dilation, which has the opposite effect. These two operations in conjunction are effective at removing noise so as to ensure that the CNN does not potentially misclassify based on irrelevant data. Finally, the resize function was used to make each image smaller (50x50), and used parameter INTER\_AREA to perform the necessary decimation. The dataset was then split into a training set and a test set using a 90/10 ratio, and the pixels were scaled by a factor of 255 so that RGB data values fell in the range of 0 to 1.

1. Convolution Neural Network (CNN)

A convolutional neural network was used to classify the images into one of the ten character classes. This involved the use of Keras, which is an API which was imported from the TensorFlow library. A sequential model was used, which is suitable for applications where there is only one input and one output to our model, which is true for our character recognition task. The CNN contains 23 layers in total, which were created sequentially using the add() function.

The model first began with two data augmentation layers: RandomContrast and RandomRotation. The purpose of these data augmentation layers is to introduce random, yet realistic transformations on the input data so as to increase diversity of the training data. RandomContrast randomly adjusted the contrast of individual color channel, whereas RandomRotation randomly rotated the images.

After data augmentation, there is a “block” of layers that we repeated three times. At the beginning of this block is two calls to Conv2D, which creates a two dimensional convolution layer. Every convolution layer used in our CNN made use of the He Normal kernel initializer, meaning it initializes the kernel matrix to have random values pulled from a variation of the normal distribution called the He Normal. The choice of kernel initializer is important as too small of values could have caused vanishing gradients and too big of values could have caused the output to diverge. Each of the convolution layers also used the rectified linear unit (ReLU) activation function. ReLU is a very popular activation function to use for CNN’s as they are computationally fast, resistant to the vanishing gradient problem, and have sparse activation [1]. After the activation function is applied, a batch normalization layer is applied. Batch normalization is a technique which is used to combat “covariate shift”, which is a problem in machine learning which leads to misclassification [2]. Batch normalization results also results in allowing for higher learning rates, which results in faster CNN [2]. MaxPooling2D follows the batch normalization, and this is a pooling layer where it down-sampled the output into a simpler image while keeping the information about the most important features. Max pooling in particular accomplished this by taking the maximum value of a pixel in a smaller sub-region. Finally, Dropout is a layer which randomly sets certain output values to zero, which is a regularization technique that helps to avoid overfitting the model.

A Flatten layer is used after the previously described block was carried out three times. It transforms the multidimensional input tensors into a single dimension representing all dimensions. Three Dense layers follow this layer. Dense layers are similar to convolutional layers in that they use an activation function and a kernel in order to produce an output image. Each dense layer has a fully connected neural network structure, and these layers are what are responsible for the final classification decisions of the entire CNN. The first two layers used the same ReLU activation function as before, and the third layer used the softmax activation function. Softmax is used here as it is a common choice for the final layer of a CNN, given that it produces an output where the sum of all outputs sum to 1. This allows us to use this output of the CNN as an input to the cross-entropy loss function, which allows for us a method for backpropagation.

1. Model Fitting and Interpretation

After all the layers have been added to the CNN, the model is then fitted with the training data , and then evaluated using the test data. We used 600 epochs for out CNN, meaning that we will go through 600 complete passes in the CNN. Evaluating the testing set returns a list of scores for each epoch, and information regarding each epoch is displayed on the screen after running the program. Finally, a graph of the cross-entropy loss as a function of the epoch is displayed.

<https://arxiv.org/pdf/1811.03378.pdf> [1]

<https://arxiv.org/pdf/1502.03167.pdf> [2]