Final Report
Implementation of a Neural Network for Multi-Class Image Classification

1. Introduction

This project focuses on building and training a neural network from scratch to classify grayscale images of alphanumeric characters into 62 classes (digits 0-9, uppercase letters A-Z, and lowercase letters a-z).

2. Dataset Preparation

2.1 Dataset Structure

The dataset is organized into 62 folders (Sample001 to Sample062):

- Sample001 to Sample010: Represent digits 0-9.
- Sample011 to Sample036: Represent uppercase letters A-Z.
- Sample037 to Sample062: Represent lowercase letters a-z.

2.2 Preprocessing

The dataset is processed using the load_and_preprocess_images function, which performs the following steps:

1. **Image Loading**: Images are loaded using the Python Imaging Library (PIL).
2. **Grayscale Conversion**: Images are converted to grayscale for simplicity.
3. **Resizing**: All images are resized to 80x60 pixels to standardize input dimensions.
4. **Normalization**: Pixel values are scaled to the range [0, 1].
5. **Flattening**: Each image is flattened into a vector of size 4800 (80x60).
6. **Label Encoding**: get_class_from_folder Maps folder names to class labels.
    - 0-9 for digits.
    - 10-35 for uppercase letters.
    - 36-61 for lowercase letters.

2.3 Data Splitting

The split_data function is used to split the dataset into training and testing sets. For each class:

- 50 random samples are selected for training.
- 5 random samples are selected for testing.

  This ensures a consistent 50:5 split while maintaining randomness.

3. Neural Network Architecture

Input Layer

- Size: 4800 neurons, corresponding to the flattened image vector.
- Input Representation: Pixel intensities scaled between [0, 1].

Hidden Layer

- Size: 100 neurons.
- Activation Function: Sigmoid.

Output Layer

- Size: 62 neurons, one for each class.
- Output Representation: A matrix of 1x62 probabilities indicating the likelihood of each class.
- Activation Function: Sigmoid.

4. Training Procedure

4.1 Initialization

The initialize_parameters function initializes weights and biases:

- **Weights**: Xavier initialization to stabilize signal flow.
- **Biases**: Initialized to 1.

4.2 Forward Propagation

The forward_propagation function computes activations for the hidden and output layers:

- Hidden layer activations are computed using the Sigmoid function.
- Output layer activations are computed using the Sigmoid function.

4.3 Loss Function

The compute_cost function calculates cross-entropy loss:

- Output activations (A2) are clipped to prevent numerical instability (log(0)).

4.4 Backward Propagation

The backward_propagation function computes gradients for weights and biases:

- Gradients are calculated for both layers using the derivative of the Sigmoid function.

4.5 Gradient Clipping

The clip_gradients function ensures gradient values remain within a specified range to prevent exploding gradients.

4.6 Hyperparameters:

- Learning Rate: 0.01
- Epochs: 2000.
- Batch size: 32

4.7 Steps Per Epoch:

- Shuffle the training data.
- Divide the training data into mini-batches.
- Perform forward propagation for each mini-batch.
- Compute the loss.
- Perform backward propagation to compute gradients.
- Update weights and biases using gradient descent in the update_parameters method.

5. Challenges and Optimizations

I faced the problem of **Vanishing Gradients** during initial training which is common when using Sigmoid as the activation function. Due to this the loss remained high (> 5) and the accuracy on the training and testing dataset was low (<3%)

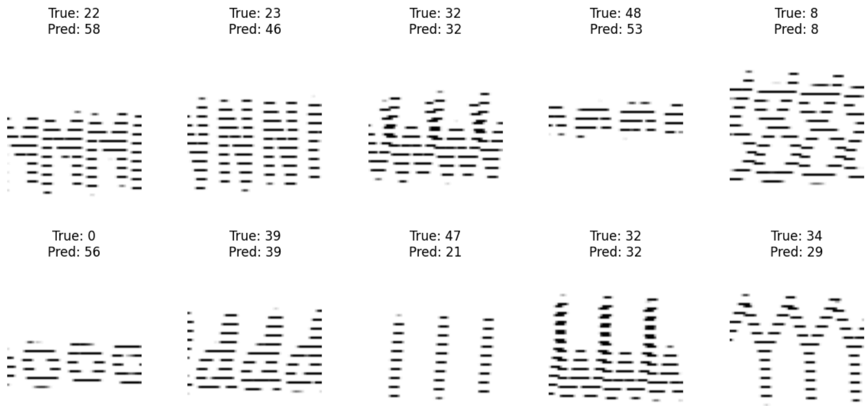
Several techniques were used to improve the model:

1. **Xavier Initialization**: Scaled weights based on input size to stabilize signal flow and prevent gradient saturation in sigmoid neurons. This ensured consistent gradient propagation and better convergence.
2. **Mini-Batch Processing**: Trained using random subsets of the dataset, stabilizing updates and improving convergence by reducing noise while maintaining stochasticity.
3. **Gradient Clipping**: Restricted the magnitude of gradients to avoid exploding gradients and ensure smoother, controlled updates.

All these measures significantly increased the accuracy of the model

6. Results

- **Training Accuracy**: 67.42%.
- **Testing Accuracy**: 43.55%.


True: 22
Pred: 58
True: 23
Pred: 46
True: 32
Pred: 32
True: 48
Pred: 53
True: 8
Pred: 8
True: 0
Pred: 56
True: 39
Pred: 39
True: 47
Pred: 21
True: 32
Pred: 32
True: 34
Pred: 29