Enhanced CNN with Self-Attention for Handwritten Digit Summation

Aryan Gupta
2021113012

Introduction

In my earlier work, I developed a preliminary Convolutional Neural Network (CNN) model to predict numerical labels (the sum of four handwritten digits) from a dataset of images. The initial approach involved normalizing the images, constructing a simple CNN architecture (with one convolutional layer and one dense output layer), and training the model using the Adam optimizer with Mean Squared Error (MSE) as the loss function.

In this final submission, we expand upon the original approach by:

- Adopting a deeper CNN architecture composed of multiple convolutional blocks.
- Introducing a self-attention mechanism to capture higher-level interactions in the feature space.
- Performing extensive hyperparameter tuning (over 22 hours) using the Keras Tuner framework (`Hyperband`) to find the best configuration of filters, kernel sizes, dropout rates, and other hyperparameters.
- Demonstrating improved performance through metrics such as training/validation loss and threshold based accuracy measures.

Preliminary Approach Recap

Previously, we aimed to predict the summed digit value using a simpler CNN framework. The dataset of handwritten digits was loaded as NumPy arrays, normalized by dividing all pixel values by 255, and split into training and validation sets. Training was carried out for 50 epochs, and the final model reached a validation loss of around 25.10, with a mean absolute error (MAE) of about 3.0 on the training data.

Enhanced Model Architecture

To build upon that foundation, we designed a more elaborate CNN architecture that includes multiple convolutional blocks followed by a self-attention module:

- **Convolutional Blocks:** Each block consists of a `Conv2D` layer with ReLU activation, Batch Normalization, and Max Pooling. Multiple blocks allow the network to learn hierarchical features from the input images.
- **Global Average Pooling:** After the final convolution block, we apply Global Average Pooling to reduce spatial dimensions while retaining salient features.
- **Self-Attention Mechanism:** We reshape the output from the CNN backbone into a suitable dimension and apply a Multi-Head Attention layer. This helps the model to attend more effectively to different parts of the learned feature map.
- **Fully Connected Layers:** A Flatten operation is followed by a dense layer with ReLU activation and a final dense output layer for regression (predicting the sum of the digits). A dropout layer is also employed to mitigate overfitting.

Hyperparameter Tuning

I utilized `Hyperband` from `keras_tuner` to systematically explore hyperparameters, including:

- Number of filters in each convolutional layer (ranges from 32 to 512)
- Kernel sizes (3 or 5)
- Number of attention heads (2 to 8)
- Key dimension for attention (16 to 64)
- Dense layer size (64 to 256)
- Dropout rate (0.2 to 0.5)
- Learning rate (between 10^{-4} and 10^{-2}, sampled logarithmically)

The tuning process took over 21–22 hours, involving multiple trials (`Hyperband` parameter `max_epochs` = 20) across different brackets. The best hyperparameters identified by the tuner were:

```
{
  'filters_1': 128,
  'kernel_1': 5,
  'filters_2': 64,
  'kernel_2': 3,
  'filters_3': 256,
  'kernel_3': 5,
  'filters_4': 128,
  'kernel_4': 3,
  'filters_5': 192,
```

```
  'kernel_5': 3,
  'num_heads': 8,
  'key_dim': 64,
  'dense_units': 192,
  'dropout': 0.3,
  'learning_rate': 0.0010331041248433069,
  ...
}
```

Implementation Details

Below is a concise outline of the key steps in my pipeline:

1. **Data Loading:** We read in all `.npy` files, concatenating them into arrays `X` (images) and `y` (labels).
2. **Preprocessing:** We ensure each image is expanded to have shape $(40, 168, 1)$, and we normalize pixel values to lie between 0 and 1.
3. **Hyperparameter Tuner Setup:** We define the model-building function `build_model` that takes a `HyperParameters` object. The `Hyperband` tuner is instantiated to search for the best combination of filters, kernel sizes, etc.
4. **Training and Early Stopping:** We train the best model found for up to 100 epochs but apply an early stopping mechanism (patience of 5 epochs) to avoid overfitting.
5. **Model Saving and Reloading:** We serialize the best model to disk using the `.keras` format. Subsequently, we reload it for additional fine-tuning or validation experiments.

Results and Observations

Training and Validation Loss

During tuning and final training, the model's training and validation loss both exhibit consistent downward trends. An example snippet of the loss curve is shown below:

- At early epochs, training loss is relatively high (e.g., around 17–18).
- Over the course of 16+ epochs, training and validation loss both settle to values around 2.5–3.0, sometimes dipping to below 2.0.

This indicates that the network is effectively learning to regress the sum of the digits. The use of a deeper CNN architecture plus attention appears to improve the representational capacity of the model, helping it converge to a lower MSE.

Threshold-Based Accuracy

Although our main objective is regression (predicting the correct sum of digits), we also track a threshold-based accuracy for interpretability:

- **0.5 cutoff:** The fraction of predictions where the absolute difference from the true label is < 0.5.
- **0.8 cutoff:** The fraction of predictions where the absolute difference from the true label is < 0.8.
- **0.9 cutoff:** The fraction of predictions where the absolute difference from the true label is < 0.9.

Why This Architecture Works

- **Deeper Convolutional Layers:** More convolutional layers can extract robust features for digit identification and summation.
- **Global Average Pooling:** This reduces the dimensionality of feature maps without excessively increasing the parameter count, preventing overfitting.
- **Self-Attention Mechanism:** The multi-head self-attention identifies important interactions among features. This can be especially helpful when subtle relationships between different digit regions matter for precise regression.
- **Hyperparameter Optimization:** Systematic exploration of parameters such as learning rate, number of filters, kernel sizes, and dropout rates ensures a tailored design that best fits the data.

Conclusion

By incorporating deeper convolutional layers, a self-attention block, and extensive hyperparameter tuning, we have significantly improved our model for predicting the summed digit values. The training and validation losses consistently fell over the course of epochs, and threshold-based accuracy measurements demonstrated robust performance.