APRIL 22, 2024

BUILDING A NEURAL NETWORK MODEL FOR A CLASSIFICATION PROBLEM

REPORT

ARMAAN IRANI

Abstract

This project focuses on exploring and tuning a multitude of hyperparameters for a multilayer perceptron neural network and a convolutional neural network. This is conducted on an image classification task using the EMNIST dataset. The primary objective of this project is to compare the efficacy of the MLP and CNN architectures with an image classification task, focusing on how the changing of hyperparameters can lead to model performance. Hyperparameters explored in this project are, optimizers, activation functions, learning rates, learning rate schedulers, layers, nodes, dropout rate, and regularization strength. The performance of each model is assessed by the accuracy metric and the impact that modifying hyperparameters have on the performance. This project not only aims to build a neural network model to solve the EMNIST classification problem, but also provide insights into the practical applications in image classification.

Introduction

Handwritten character recognition is one of the fundamental problems of computer vision and pattern recognition with applications extending to OCR (Optical Character Recognition), document processing, and digitization of forms. The ability to accurately classify handwritten characters is crucial for automating numerous data entry and processing tasks, hence improving efficiency and reducing human effort.

This task aims at building neural network models for recognizing characters written by hand from the EMNIST balanced (Extended MNIST) dataset. EMNIST is an extended MNIST dataset of handwritten letters, which further extends diversity and size beyond that achieved by the original. The EMNIST dataset is a derivative of the NIST Special Database 19 and, therefore, is structured in the same way the iconic MNIST dataset is. The current project has investigated two types of architectures in neural networks: Multilayer Perceptron (MLP) and Convolutional Neural Networks (CNNs). MLPs which are composed of multiple fully connected layers which are capable of learning complex, non-linear mappings between input and the outputs. On the other hand, CNNs are designed for grid-like data, such as images, exploiting local connectivity patterns and translation invariance which makes them better suited for computer vision tasks.

The objectives of the project are:

1. Implementing and training both MLP and CNN models for the task of recognizing characters present in EMNIST.
2. Try hyperparameters such as activation functions, optimizers, regularization methods, learning rate schedulers, etc., which can help improve the model performance and generalization ability.
3. Compare the performance of the MLP and CNN models concerning their strengths, weaknesses, and the suitability of each for the task at hand.
4. Approach the process of model building, regularization techniques to deal with overfitting and underfitting, and factors that affect the performance of a neural network model more intelligently.

The following libraries were used for this project:

1. Pandas: Data manipulation and analysis
2. NumPy: Support for large, multi-dimensional arrays and matrices.
3. Matplotlib: Providing plots and figure visualization.
4. Seaborn: Providing statistical graphs like the confusion matrix.
5. Time: Calculating training time of each model.
6. Scikit-learn: splitting the data and other evaluation metrics for the models.
7. TensorFlow: Creating MLP and CNN models using Keras.
8. Keras-Tuner: Hyperparameter optimization framework.

The functionality and usage of each library will further be discussed in the report.

In the process, this project would contribute to the present ongoing research and development efforts in the area of handwritten character recognition while providing hands-on practical experience in the development, training, and evaluation of a neural network model for such a real-world classification task.

Methodology

2.1 Data preprocessing

This subsection will describe the steps taken to prepare the EMNIST dataset for use with the neural network models. Key preprocessing steps include:

1. Data Loading: How the data is loaded into the environment using custom helper functions.
2. Normalization: The process of scaling pixel values to a range of 0 to 1 to facilitate faster and more stable training.
3. Train-Val Split: The criteria and method used to divide the dataset into training-validation sets.

A custom helper function was designed specifically for the task of loading the data using the pandas read_csv function. The features and labels were separated into X and Y values using the NumPy iloc method. Normalization of the X values is also done within this function by dividing each value in the NumPy array by 255 which is the largest value in the array. This is done in order to have all the values in the array to be in the range of (0,1). This improves the performance and training stability of the model.

2.2 Data structure and visualization

To understand the shape and number of values in each set, the shape of each data is printed. Using the custom helper function created for visualizing the data incorporates the meticulous loop going through each image and plotting them with their corresponding labels provided in the mapping file using matplotlib. I plotted 10 images to understand them. The images provided were disoriented, so the usage of another custom helper function which rotated each image by 90 degrees clockwise and the flip them horizontally, helped resolve that issue.

A further note to add, the data had to be reshaped to (28, 28, 1) in order to be fed into the convolutional layers. Every feature data was reshaped by using the NumPy reshape function.

2.3 Model Development

Multilayer Perceptron (MLP):

The initial baseline model was set up having 3 hidden layers, the first layer having 512 nodes, second layer 1024 nodes, and the third layer 1024 nodes using the ReLU activation function as the default for each layer. The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time. The output layer consisted of the number of classes (47) with the SoftMax activation due to the nature of it being a multiclass classification task. Stochastic gradient descent was the optimizer used. This configuration produced an accuracy of 69.45% when tested on the test data.

Convolutional Neural Network (CNN):

The initial model configuration for the CNN has two Convolutional layers, each followed by a pooling layer with the kernel size of (2,2). The first and second convolutional layers consists of 32 and 64 filters respectively. The data was flattened to be then fed to a fully connected dense layer having 128 nodes. ReLU was the initial choice of activation function throughout the model except the output layer being SoftMax due to its multiclass classification nature. The model was then compiled using the Adam optimizer.

2.4 Hyperparameter tuning

The section below will briefly describe the methodologies applied for model hyperparameter tuning used in optimization:

Hyperparameters explored and tuned:

1. Learning rate schedulers: Two different learning rate scheduling methods, Exponential decay and Inverse Time decay were implemented to adjust the learning rate during training and potentially improve convergence.
2. Learning rate: A range of learning rates were tested during training.
3. Activation function: ReLU, Leaky ReLU, and ELU were experimented with to find the most suitable non-linear activation for each model.
4. Optimizers: SGD, Adam, and RMSprop were evaluated to identify the optimizer that achieved the fastest convergence and best performance.
5. Number of layers and nodes: Number of layers and nodes in each layer were tried and tested.
6. Dropout: Dropout was employed to prevent overfitting by randomly dropping out a fraction of neurons during training.
7. Regularization: L1 and L2 regularization were applied to the models to mitigate overfitting.

Tuning Technique:

Keras Tuner was the choice of optimization framework as it was easily integrable with the Keras models and helps with the search for different hyperparameters. Two tuning techniques were used viz. random search and Bayesian optimization. Bayesian optimization builds a probability model of the objective function and use is to select the hyperparameters to evaluate the true objective function. Unlike random search and grid search, Bayesian optimization takes into account the performance of past hyperparameters to help make the decision in selecting the future ones. Hence, this algorithm is a much more efficient method

of searching through large combinations of parameters such as the number of layers and nodes, learning rate, and dropout rate.

Values of the hyperparameters and methods were systematically tuned. A model was created for the baseline configuration with some accumulated knowledge and default values. Then, each hyperparameter or technique was fine-tuned in isolation, keeping all other settings of the previous iteration at their best identified configuration. This implies that all other settings are kept at the best identified throughout this iterative process, which finally enabled the identification of the best combination of hyperparameters and techniques for each model.

2.5 Training the Models

2.5.1 Batch Size and Epochs

A batch size of 32 was used to train the models. Other explorations of lowering and increasing the batch size were made, but 32 proved to have the best train time vs accuracy trade off among batch sizes of 16 where it took twice the time to train, but had a slight increase in accuracy and batch size of 64, where the training was completed in half the time, although a slight sacrifice was made on the accuracy. Furthermore, the models were trained on 10 epochs regularly, 5 epochs when using the Keras Tuner to help reduce the time, and 25 on the final models along with early stopping.

2.5.2 Optimization Algorithms

The Tuner compared the performance between SGD, Adam, and RMSprop, making Adam the choice of optimizer for both, the MLP and CNN model. As Adam is an adaptive learning rate algorithm designed to improve training speeds and reach convergence quickly which made it proved to be the ideal choice.

2.5.3 Performance Metrics

Accuracy was the metric monitored throughout the training process, making it the reason of distinguishing different models. More importantly, the accuracy on the validation set was monitored and compared with the training accuracy for the search of overfitting or underfitting. The models were tending to overfit in all cases, the steps taken to mitigate overfitting will be discussed below.

2.5.4 Handling overfitting

In the stages of model development and search for optimal model performance, the addition of batch normalization layers, dropout layers, and regularization were explored and tuned with the help of the Keras Tuner. Batch normalization helps increase the speed of convergence and providing stability in neural networks by providing an additional layer. Knowling this, the Tuner cycled to understand which combination of including batch normalization helped improve the performance. Similarly, introducing dropout in a layer randomly sets the input of nodes to zero given with a probability known as the dropout rate. Using the Bayesian algorithm for this ensures the proper exploration for each of these probabilities. Finally, the use of regularization helped with the generalization of the model which is done by introducing a penalty term to the loss function during training. Both L1 and L2 regularizers were explored by the Tuner, again using the Bayesian Optimization.

2.6 Model Evaluation

The performance of the MLP and CNN was evaluated against a number of metrics calculated over the test set: accuracy, precision, recall, and F1-score. A confusion matrix was plotted using seaborn to check the correlation between true and predicted values. Finally, an accuracy and loss graph of the model over the number of epochs between the training and validation set were plotted helping to understand the model's performance.

Comparative Analysis of the MLP and CNN Models

CNNs are designed explicitly for making use of the spatial and local connectivity patterns of image data, hence are very suitable for tasks in visual data, such as handwritten character recognition. However, the convolutional layers work well in CNN to grab low-level features, such as edges and shapes, so after many layers, high-level features or representations are learned, and hence the model is ready to recognize complex patterns or characters.

On the other hand, MLPs interpret the input image as a flattened vector, and may discard essential spatial information. Although MLPs are able to learn complex non-linear mappings well, they fail to capture the intricate patterns and details present in handwritten characters most of the time, hence performing poorly in comparison to CNNs. The performance of these two models is, however, dependent on the dataset, and a host of other factors which include how complex the dataset is, the availability of training data, and the exact characteristics in relation to the task at hand. Moreover, in some particular cases, for example dealing with lesser data samples or simple data samples, MLPs can be more effective in that they need a reduced computational burden and form a simpler architecture.

Conclusion

This work aimed to develop and further evaluate a neural network model on the task of character recognition from handwritten examples given by the EMNIST balanced dataset. These included two types of models: a Multilayer Perceptron (MLP) and a Convolutional Neural Network (CNN) applied through designed, developed, and finely tuned techniques of hyperparameter optimization. The results herein show CNN model superiority in accuracy, precision, recall, and f1-score over the MLP model. CNN achieved an accuracy of 86.15% on the test set, while MLP recorded 86.89%.

Hyperparameter tuning contributed to reaching successful convergence, generalization, and overall accuracy of the models. The results of this study can, therefore, be used in a real environment for different applications like character recognition, document processing, or OCR. Superior performance of the CNN model would, therefore, warrant deployment in those cases where high accuracy and robustness are called for in real-world scenarios. However, here we must note that the performance of such neural network models depends heavily on both the task difficulty and the amount of training data that can be gathered, along with the characteristics of each specific problem domain. Under some simple problem conditions or small data-set sizes, some practitioners prefer the use of MLPs due to their simpler architecture and lower computational requirements.

References

Pandas - python data analysis library (no date). Available at: https://pandas.pydata.org/ (Accessed: 21 April 2024).

Harris, C.R. et al. (2020) 'Array programming with NumPy', Nature, 585(7825), pp. 357–362. Available at: https://doi.org/10.1038/s41586-020-2649-2.

The Matplotlib Development Team (2024) "Matplotlib: Visualization with Python". Zenodo. doi: 10.5281/zenodo.10951225.

Waskom, M. (2021) 'Seaborn: statistical data visualization', Journal of Open Source Software, 6(60), p. 3021. Available at: https://doi.org/10.21105/joss.03021.

Pedregosa, F. et al. (2012) 'Scikit-learn: machine learning in python'. Available at: https://doi.org/10.48550/ARXIV.1201.0490.

Tensorflow (no date) TensorFlow. Available at: https://www.tensorflow.org/ (Accessed: 21 April 2024).

Team, K. (no date) Keras documentation: KerasTuner. Available at: https://keras.io/keras_tuner/ (Accessed: 21 April 2024).

Jain, S. jain (2018) 'An overview of regularization in deep learning(With python code)', Analytics Vidhya, 19 April. Available at: https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/ (Accessed: 21 April 2024).

Saxena, S. (2021) 'Introduction to batch normalization', Analytics Vidhya, 9 March. Available at: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-batch-normalization/ (Accessed: 21 April 2024).

Complete guide to the adam optimization algorithm | built in (no date). Available at: https://builtin.com/machine-learning/adam-optimization (Accessed: 21 April 2024).

Gupta, D. (2020) 'Fundamentals of deep learning - activation functions and when to use them?', Analytics Vidhya, 29 January. Available at: https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/ (Accessed: 21 April 2024).