Digit Recognizer (Kaggle)

Ritesh Kumar

2024WI_MS_DSP_422-DL_SEC61: Practical Machine Learning

Module 7 Assignment

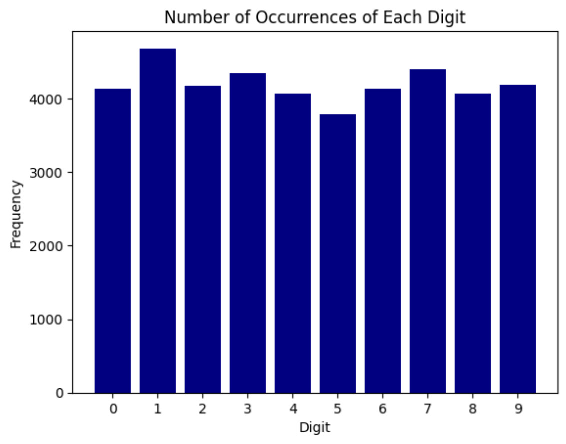
Digit Recognizer

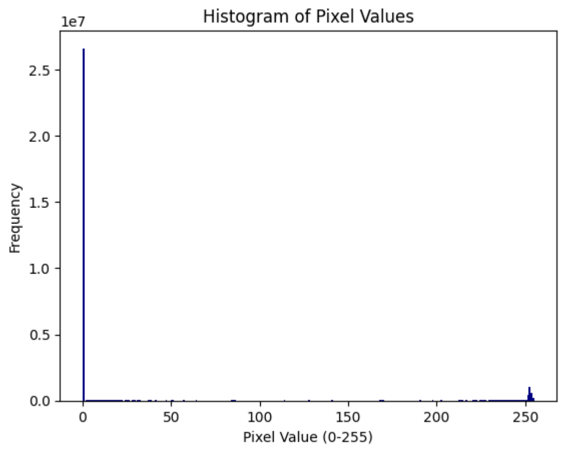
Donald Wedding and Narayana Darapaneni

February 16, 2024

Exploration

We start with a comprehensive Exploratory Data Analysis (EDA), commencing with the extraction of descriptive statistics for the ‘label column, serving as the target variable for prediction. We also plot the pixel-distribution for the 42000 records.

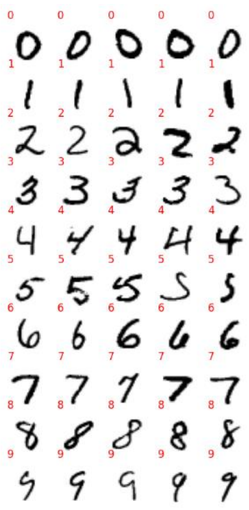




The distribution of digits is roughly uniform, suggesting a balanced dataset which is ideal for machine learning classification tasks, as it may prevent bias towards more frequent classes.

The pronounced peak at the left indicates an abundance of black pixels, while the peak at the right suggests a presence of white pixels. The scarcity of mid-range values implies these images have high contrast with predominantly black backgrounds and lighter features i.e. digits are typically white on a black background, contributing to the stark bimodal distribution observed.

We displayed a sample of the digits.



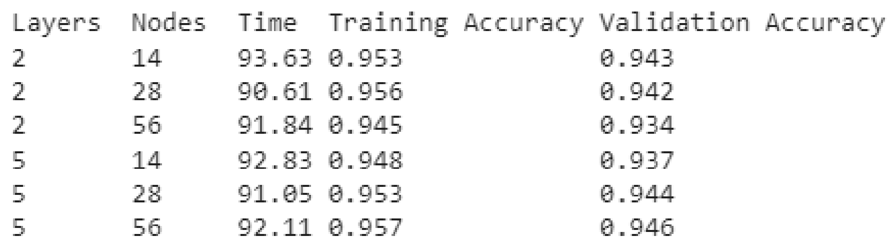
Model Development

Design of Experiments was conducted by development and evaluation of models using TensorFlow and Keras, encompassing a comprehensive approach to machine learning tasks, from data preparation to model optimization and evaluation. Various architectures were explored, including Dense Neural Networks (DNNs), Convolutional Neural Networks (CNNs), and Long Short-Term Memory (LSTM) networks, each tailored to the specific nature of the data. For instance, DNNs are optimized through grid searches focusing on hyperparameters like the number of layers and neurons, leveraging GridSearchCV for systematic tuning to improve accuracy. Similarly, CNNs are designed with considerations for the number of convolutional layers and filters, capitalizing on their ability to extract spatial features from image data. LSTMs, known for their efficiency in handling sequence data, are optimized by adjusting layers and units to capture temporal dependencies effectively.

The evaluation of these models involves not just examining training and validation accuracies but also employing techniques like early stopping to prevent overfitting. The results indicate that the optimal configurations vary by model type, emphasizing the importance of architectural design in achieving high performance. The process underscores the iterative nature of machine learning, where data exploration, model building, and rigorous evaluation converge to guide the optimization of neural network architectures for enhanced predictive capabilities.

Model Deployment

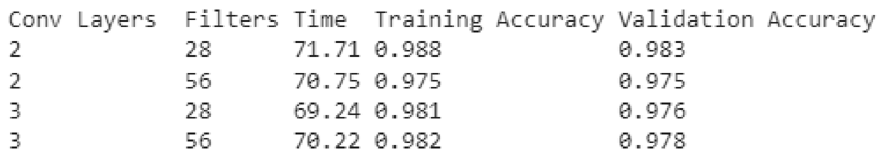
Dense Neural Network: The code outlined a method for optimizing a neural network's structure via grid search, focusing on hyperparameters like the number of layers and neurons. Using the build_model function, it builds models with different configurations, evaluated against a dataset split into training and validation sets. This approach allows for exhaustive exploration of parameter combinations to identify the most effective neural network architecture. The process employs the Keras library for model construction and optimization, leveraging the GridSearchCV class from Scikit-Learn for systematic hyperparameter tuning, aiming to maximize accuracy. Results were collected and displayed, highlighting the impact of different configurations on model performance.



```
Layers  Nodes  Time  Training Accuracy Validation Accuracy
2       14     93.63 0.953             0.943
2       28     90.61 0.956             0.942
2       56     91.84 0.945             0.934
5       14     92.83 0.948             0.937
5       28     91.05 0.953             0.944
5       56     92.11 0.957             0.946
```

The search tested combinations of 2 and 5 layers with 14, 28, and 56 nodes, assessing their impact on model performance over time, and evaluating training and validation accuracy. Models with 5 layers and 56 nodes achieved the highest validation accuracy (0.946) and training accuracy (0.957), indicating a potential sweet spot in complexity for this particular dataset. Conversely, models with 2 layers and 56 nodes showed lower validation accuracy (0.934), suggesting that simply increasing nodes without adjusting layers might not always yield better results.

Convolutional Neural Network (CNN): The code introduced an approach to optimize a Convolutional Neural Network (CNN) model for image classification tasks. It involves defining a function, build_cnn_model, which constructs a CNN with customizable parameters such as the number of convolutional layers, number of filters, kernel size, learning rate, and input shape. The model aims to maximize accuracy through iterative training and validation, utilizing Keras for model construction and Adam for optimization. A grid search strategy is employed to systematically explore different configurations of convolutional layers and filters, facilitated by the GridSearchCV tool from Scikit-Learn, with the objective of finding the optimal model settings for enhanced performance on a given dataset.

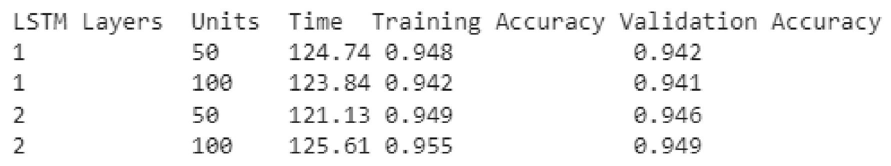


```
Conv Layers  Filters Time  Training Accuracy Validation Accuracy
2            28      71.71 0.988             0.983
2            56      70.75 0.975             0.975
3            28      69.24 0.981             0.976
3            56      70.22 0.982             0.978
```

The summary details the outcomes of optimizing a Convolutional Neural Network (CNN) through grid search, varying the number of convolutional layers and filters. The configurations explored include 2 or 3 convolutional layers with 28 or 56 filters. The best model, with 2 convolutional layers and 28 filters, achieved the highest training accuracy of 98.8% and validation accuracy of 98.3%, indicating superior performance and generalizability. Models with more filters or layers did not necessarily perform better, highlighting the importance of finding the right balance between model complexity and efficiency for optimal performance.

Long Short-Term Memory (LSTM): We designed and optimized a Long Short-Term Memory (LSTM) neural network model for sequence data processing.

The build_lstm_model function is central to this process, enabling the creation of models with varying numbers of LSTM layers and units, tailored to the specific structure of the input data. By leveraging a grid search technique, the code aims to find the optimal combination of LSTM layers and units that maximizes model accuracy. This approach is facilitated by Keras for model construction and optimization, and the search for the best hyperparameters is conducted using Scikit-Learn's GridSearchCV, focusing on maximizing accuracy. The process not only involves model training and validation but also a thorough evaluation of the results to identify the configuration that offers the best performance on the given dataset.



```
LSTM Layers  Units  Time   Training Accuracy Validation Accuracy
1            50     124.74 0.948             0.942
1            100    123.84 0.942             0.941
2            50     121.13 0.949             0.946
2            100    125.61 0.955             0.949
```

The summary showcases the results of optimizing a Long Short-Term Memory (LSTM) network through varying the number of layers and units. Models were assessed based on their training and validation accuracies. The configuration with 2 LSTM layers and 100 units emerged as the most effective, achieving the highest training accuracy of 95.5% and validation accuracy of 94.9%. This indicates that increasing both the number of layers and the complexity (units) of the LSTM model can lead to improved performance on the dataset, suggesting a better capability to capture complex patterns and dependencies in the data.

Conclusion

Analyzing the outcomes from experiments with traditional densely connected Neural Networks alongside those of Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks provides a comprehensive view of model optimization across different data types and architectural paradigms. The densely connected network optimization highlighted the effectiveness of deeper models with more neurons, particularly showing that a configuration with 5 layers and 56 neurons outperformed others in terms of validation accuracy. This finding aligns with the broader theme observed across the experiments: the importance of tailoring the network architecture to the task at hand for optimal performance.

The CNNs proved most adept at image processing tasks, optimizing at a relatively simpler architecture with 2 layers and 28 filters, emphasizing the value of spatial feature extraction in image data. In contrast, the LSTMs, with their ability to process sequential data, reached peak performance with 2 layers and 100 units, showcasing their strength in capturing temporal dependencies.

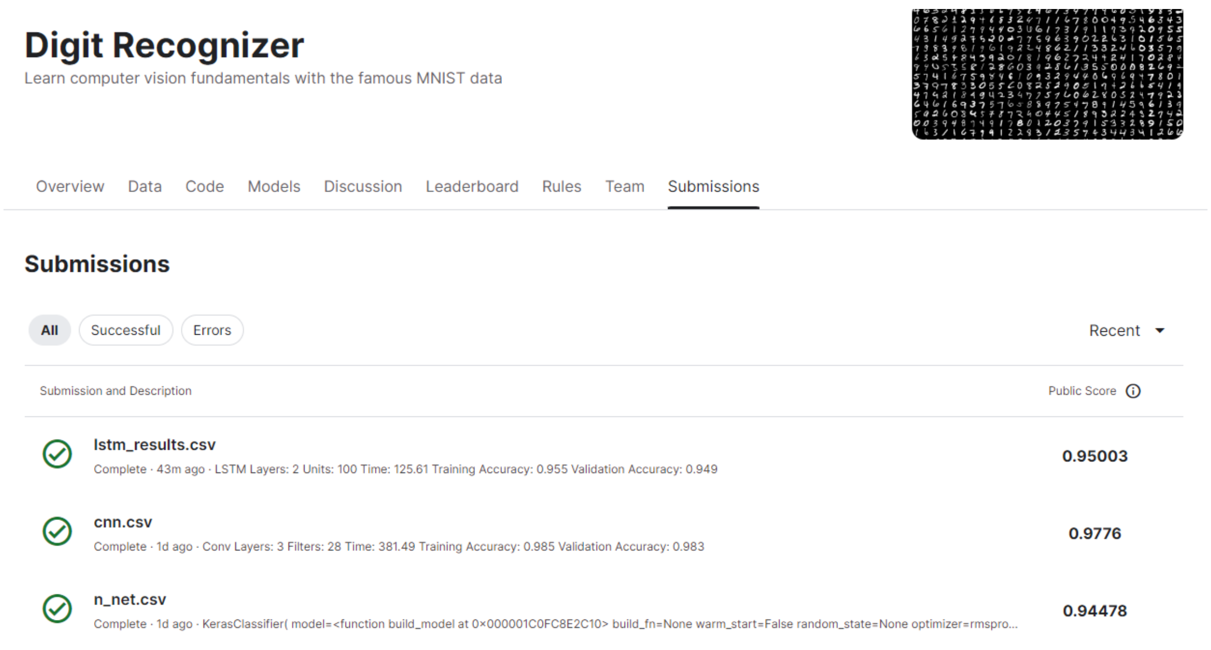
Across all models, the grid search strategy was essential in navigating the complex landscape of hyperparameters to identify configurations that maximize accuracy. This exploration underscores the critical role of architectural design in achieving high performance, demonstrating that while there is no universal solution, certain patterns—such as the depth of the network and the number of processing units—consistently influence outcomes. These insights reinforce the principle that effective model optimization is contingent on understanding the unique characteristics of the dataset and task, guiding the selection of the most suitable neural network architecture.

Kaggle Submission

My Kaggle username is riteshrk.

The notebook has been uploaded on Kaggle and can be accessed here.

The results of all the three models were submitted on Kaggle and can be accessed here.



Digit Recognizer

Learn computer vision fundamentals with the famous MNIST data

Submissions

Submission and Description	Public Score
lstm_results.csv Complete · 43m ago · LSTM Layers: 2 Units: 100 Time: 125.61 Training Accuracy: 0.955 Validation Accuracy: 0.949	0.95003
cnn.csv Complete · 1d ago · Conv Layers: 3 Filters: 28 Time: 381.49 Training Accuracy: 0.985 Validation Accuracy: 0.983	0.9776
n_net.csv Complete · 1d ago · KerasClassifier(model=<function build_model at 0×000001C0FC8E2C10> build_fn=None warm_start=False random_state=None optimizer=rmspro...	0.94478

For the test data on Kaggle, the best accuracy of 97.76% was returned for the Convolutional Neural Networks Model.

Code