Comparison of Various Machine Learning Models for Handwritten Character Recognition

Gabrielle Chavez, Jackson Shapiro, Mehmet Emre Tiryaki

***Abstract*—Handwritten text continues to play a vital role in an increasingly digitized world, demanding effective methods for converting handwritten content into digital formats [1]. This project focuses on classifying handwritten text into its corresponding ASCII representations. To achieve this, we evaluated a range of traditional machine learning and deep learning models. Our findings indicate that a transformer model augmented with convolutional neural network (CNN) features delivers the highest performance, achieving an accuracy of 86%. These results highlight the potential of combining self-attention mechanisms with CNNs for robust and accurate handwritten text classification.**

***Index Terms*—Random Forest, K-Nearest Neighbors, Extreme Gradient Boosting, Feed-Foward Neural Network, CNN, Transformer, OCR.**

I. Introduction

The world is progressing into a digital era, where humans interact with technology in all aspects of their lives, including medicine, transportation, and education [2]. In addition, daily tasks are becoming increasingly integrated with technological solutions— like handwriting. The importance of accurately and automatically transcribing handwritten documents into ASCII code continues to grow. For instance, handwriting recognition is important to process and classify documents, create digital notes, and for data entry [3]. Our project focuses transcribing handwritten characters into digital text using a range of techniques including traditional machine learning models and neural networks, while comparing the methods.

II. Data

We use the **AlphaNum** dataset [4]; it is composed of 108,791 grayscale images of handwritten characters that are either 24x24 or 28x28 pixels. The dataset is divided into training, testing, and validation subsets. Each image is labeled with the corresponding character's ASCII code.

The dataset also has a "null" class comprised of images generated of random noise to create randomly scattered pixels of gray shades on a white background. The purpose of this class is to help the models disregard the non-character parts of the images with handwritten characters.

III. Methods

In this section we describe the traditional and deep learning models.

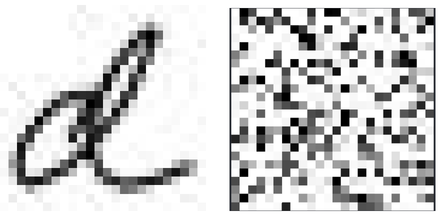

Fig. 1: Example images, a handwritten "d" character on the left and a "null" image on the right

A. Traditional Machine Learning Models

We evaluate the following traditional machine learning models:

- **Random Forest (RF)** [5]: An ensemble method based on decision trees. It aggregates multiple tree predictions to improve classification accuracy. A number of decision trees are generated and their votes are aggregated. The implementation in this project used the CART algorithm to generate decision trees. At each node a binary split is created based on the highest information gain among possible splits. The highest information gain criterion is the least log loss. The split continues until a stopping point is reached. The stopping point can be whether max depth has been reached or all the features has been used for splitting. Each tree is generated on a random subset of the data using bootstrap as to promote diversity among trees. Then, in the prediction step each tree classifies the point and the overall classification is the class that gets the most votes. Through Bayesian Optimization, we arrived at an optimal forest size of 184 trees, with each
- **K-Nearest Neighbors (KNN)**: A simple yet effective instance-based learning algorithm that assigns a class based on the majority vote of its nearest neighbors. In our model, we converted each image into a vector and used the formula $\frac{1}{\sqrt{\sum(I_{test}-I_{train})^2}}$ to find the weighted distance [6].
- **Extreme Gradient Boosting (XGBoost)** [7]: This is a model known for its efficiency and accuracy in gradient boosting, iteratively refines decision trees to minimize loss and enhance classification performance. To optimize the model, we used Bayesian optimization, identifying the ideal configuration as 140 trees with a maximum depth of 8 per tree. Additionally, we employed cross-entropy loss paired with L2 regularization, effectively reducing overfitting while maintaining robust predictive

performance.

B. Deep Learning Models

We also explore several deep learning architectures:

- **Feed-Forward Neural Network (FFNN)**: A fully connected network serves as a baseline for learning character features. The architecture comprises three hidden layers, with each successive layer reducing in size by half. Each layer applies a ReLU activation function to process the features effectively to produce logits.

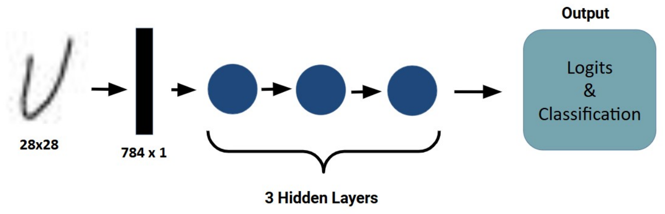

Fig. 2: FFNN Architecture

- **Convolutional Neural Network (CNN)** [8]: This deep learning model is designed to effectively capture spatial hierarchies within image data. The CNN consists of two convolutional layers: the first layer processes a single input channel and produces four output channels, while the second layer takes these four channels as input and outputs eight channels. Both layers utilize a kernel size of 3 with a stride and padding of 1, ensuring the preservation of spatial dimensions. Additionally, max pooling is applied with a kernel size of 2 and a stride of 2 to reduce spatial dimensions, enhancing feature extraction efficiency. After the second layer, it is then flattened and using a ReLU activation function it gathers logits for classification.

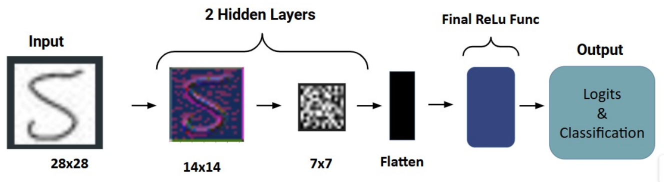

Fig. 3: CNN Architecture.

- **Transformer with CNN Features** [9]: This advanced architecture combines the strengths of transformers for capturing global context with CNNs for efficient feature extraction. In this model, we utilize the same CNN architecture described earlier, but instead of producing a final prediction, it outputs a compressed spatial feature map. This feature map is subsequently passed to the transformer, which is designed with three encoder and decoder layers that each have eight attention heads. Lastly, after the final decoder we use a linear layer to convert the information in logits for classification.

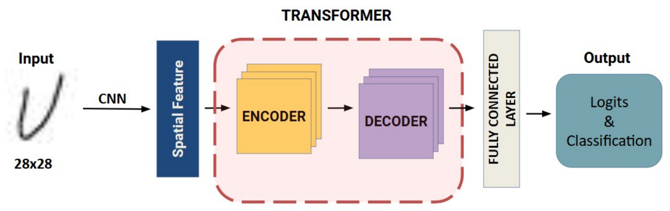

Fig. 4: Transformer

1) Training: All experiments are trained with the following parameters:

- Number of Epochs: 20
- Batch Size: 64
- Learning Rate: 0.001

In each epoch, the accuracy and loss are computed for both the training and validation sets. The validation set is included to evaluate the model's performance and mitigate the risk of overfitting. Early stopping is employed if the validation loss fails to improve after three consecutive iterations, effectively halting the training process. Additionally, a learning rate scheduler is used during each iteration to dynamically adjust the learning rate. Once training is complete, the model undergoes testing to assess its final performance.

2) Testing: To assess the model's performance, we use the AlphaNum test dataset and gather the accuracy, average loss, f1 score, precision, and recall.

C. Hyperparameter Tuning

In order to optimally select hyperparameters for the Random Forest, KNN, and XGBoost models, we utilized Bayesian optimization. This approach takes advantage of a probabilistic model to converge on collections of parameters that will likely improve performance. Within the Bayesian search model, k-fold validation with five folds prevents overfitting by testing on $\frac{1}{5}$ of the data in each iteration.

A search space of possible hyperparameter values is initialized at the beginning. Initial points of hyperparameter values in the search space are randomly selected. Using these hyperparameter values, the model is trained. Each model is evaluated using 5-folds cross-validation. Then a Gaussian Process is built as a model to approximate the validation loss as a function of hyperparameters. The next point to train a model on is then selected by choosing the point that maximizes the expected improvement. The process continues until the pre-determined last iteration value.

1) Random Forest:

- Number of Trees: 497
- Max Depth per Tree: 41
- Min Split Samples: 6
- Information Gain Criterion: Log Loss
- Weight: Balanced

2) XGBoost:

- Number of Trees: 200
- Max Depth per Tree: 8

3) KNN:

- Number of Neighbors: 3

For the hyperparameters of the neural networks, Bayesian Optimization is not a realistic choice due to the long training times for each model. We have experimented with different hyperparameter values while keeping track of validation loss, gradients,improvement between epochs and time taken to train an epoch to determine the final hyperparameter values manually. The values can be seen in the training section of the report.

We have used scikit-learn [10] and PyTorch [11] to train and test the models.

IV. Results

After training and testing each model, we wanted to compare them using the following:

- **F1-Score:** Computes average between precision and recall
- **Accuracy:** Measures how often the model predicts the correct outcome: $A = \frac{\text{Correct Prediction}}{\text{Total Predictions}}$
- **Precision:** Measures how often a model correctly classifies an image [12]. This follows the formula $P = \frac{TruePositives}{\text{True Positives + False Positives}}$
- **Recall:** Measures how often a model correctly identifies a particular image and how often it misidentifies it. It is calculated using the formula: $R = \frac{TruePositives}{\text{True Positives + False Negatives}}$

For the precision, recall, and F-1 values the weighted mean of the classes based on their abundance in the dataset was used as this is a multi-class classification problem.

Model	F1-Score	Accuracy	Precision	Recall
XGBoost	0.742	0.745	0.746	0.745
KNN	0.729	0.726	0.744	0.726
Random Forest	0.748	0.754	0.763	0.754
FFNN	0.732	0.740	0.747	0.734
CNN	0.829	0.831	0.837	0.831
Transformer	**0.863**	**0.863**	**0.874**	**0.863**

TABLE I: Results for each model

Model	Time Taken to Train (Minutes)
XGBoost	25.6
KNN	0.7
Random Forest	8.8
FFNN	6.4
CNN	7.7
Transformer	49.1

TABLE II: Training times for each model

The transformer model had the highest evaluation metrics possible out of all of them. A correlation between F-1 Score and Accuracy can also be observed. CNN as a classifier for images is a widely used model, and in the results the effectiveness of it can be seen compared to other models. The self-attention mechanism that transformer brings on top of the CNN has also proved to improve accuracy.

An important thing to note is that the traditional models are in-line with the feed-forward neural network and are relatively close with the CNN and the Transformer. This shows how they can still be viewed as valid model choices on these classification tasks. This is especially true with the trade-off of training time to accuracy being low using the KNN model.

Initially, we expected the training times were expected to be higher for the neural networks than the traditional models. However as seen in table II neural network models can be faster than traditional models. In fact, the two neural-network models FFNN and CNN took less time to train than the Random Forest and XGBoost model. This is affected by the relatively small scale of the dataset used and the hyperparameters but is still a good implication on how simple neural network architectures can be leveraged on small tasks without an expectation of really long training times.

V. Challenges

During the coding project, several challenges were encountered. One of the main issues was with the data, where we found inconsistent image sizes and incompatible class labels created difficulties in preprocessing and model training. For instance, the class labels began at '33' instead of '0'. To fix this issue, we created a mapping that would preserve the original label while assigning a valid input for the neural networks.

Training the models proved to be another hurdle, as access was limited to CPUs, making the training process slower, and all models required significant time to train. For instance, the KNN model was able to complete training in 39 seconds while transformer took 49 mins and 12 seconds.

Additionally, hyperparameter tuning became a challenge when using Bayesian Optimization, as it was too computationally intensive for the available resources. These obstacles required careful consideration and adaptation throughout the project.

VI. Conclusion and Future Work

In conclusion, this project demonstrated the strengths and trade-offs of various machine learning models for transcribing handwritten text into ASCII code. The transformer model emerged as the most effective, being able to achieve the highest evaluation metrics, while the CNN proved its reliability as a robust image classifier. Traditional models, despite their simplicity, showcased competitive performance, highlighting their continued relevance for classification tasks, especially when considering the balance between training time and accuracy. Interestingly, the results revealed that neural network models like FFNN and CNN can train faster than traditional models on smaller datasets, challenging the assumption that neural networks always require extensive computational resources. These findings underscore the potential of leveraging both traditional and neural network approaches based on the specific needs of a task, making informed trade-offs between accuracy, efficiency, and computational demands.

The work can be extended mainly by improving the task from individual character transcription to word- or sentence-level transcriptions. The Transformer model can also be improved upon by careful addition of more encoding blocks and layers. Various novel models can be tested and compared.

References

[1] P. b. A. Writer, “Handwriting transcription in the digital age,” Mar 2024.

[2] J. Wolff, “How is technology changing the world, and how should the world change technology? — global perspectives — university of california press.”

[3] Super.ai, “Handwriting recognition powered by artificial intelligence,” Jan 2023.

[4] L. Rauml;disch, “Alphanum,” Aug 2023.

[5] L. Breiman, 2001.

[6] A. AI, “Deep dive on knn: Understanding and implementing the k-nearest neighbors algorithm,” Jun 2023.

[7] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” 2016.

[8] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, “Review of deep learning: concepts, cnn architectures, challenges, applications, future directions,” *Journal of Big Data*, vol. 8, Mar. 2021.

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” 2017.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and Édouard Duchesnay, “Scikit-learn: Machine learning in python,” 2011.

[11] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, and et al., “Pytorch: An imperative style, high-performance deep learning library,” Dec 2019.

[12] E. Team, “Accuracy vs. precision vs. recall in machine learning: What’s the difference?.”