A Survey of Classification Algorithms Applied to Handwritten Character Recognition

Jesse Jones
sch. Natural Resources & Environment
University of Florida
Gainesville, Florida, United States
jcj9898@ufl.edu

Stephen Wormald
dept. Electrical & Computer Engineering
University of Florida
Gainesville, Florida, United States
stephen.wormald@ufl.edu

Mark Yen
dept. Mech. & Aerospace Engineering
University of Florida
Gainesville, Florida, United States
markyen@ufl.edu

***Abstract*—There are numerous machine learning algorithms appropriate for classification tasks, each with varying assumptions and complexity. This paper applies multiple models to the problem of handwritten character recognition as a project topic for the Fundamentals of Machine Learning course at the University of Florida. Convolutional neural networks performed the best in initial tests, and were down-selected for continued hyperparameter optimization until achieving a test accuracy greater than 90% for a set of ten characters. A second model was developed to identify filter images that were not in original character set, resulting in a filter accuracy greater than 95%.**

I. Introduction

IMAGE recognition is a longstanding pursuit of research and algorithm development in the field of Machine Learning (ML). Handwritten character recognition often serves as a standardized, computationally-simple benchmark for comparing performance across various model types [1]. The perceptron and ADALINE are landmark models that attempt to solve the character identification problem [2, 3], providing example of some of the earliest artificial Neural Networks (NN). However, these are just two of many models in a long pedigree of algorithms developed for classification.

Aside from the perceptron, there are many classification approaches. Linear classifiers include Support Vector Machines (SVM) and Linear Discriminant Analysis (LDA). Probabilistic models include the Naive Bayes (NB) classifier and Logistic Regression (LGA). There are also non-parametric models, such as *k*-Nearest Neighbors (*k*-NN) and weighted-*k*-NN. Connectionist models such as NNs took biological inspiration from the brain by implementing interconnected neuronal architectures that provided a mathematical basis for learning complex patterns in high-dimensional feature spaces [4]. While NNs became useful in image classification, the evolution of Convolutional Neural Networks (CNNs) further improved classification performance for high dimensional data such as images by encoding spatial information in the NN architecture [5]. There is rich history in the development of classification algorithms, but it can be challenging to appreciate the change in model performance over time.

This paper demonstrates the performance of various algorithms when applied to the image recognition of handwritten characters (image set ϕ), and is a final project submission for the Fundamentals of Machine Learning course at the University of Florida (EEL5840). A key grading criteria was classification accuracy, so we compared the performance of multiple model types (Section III-A) and down-selected the best found type for fine-tuning and submission in the project (Section III-B). After tuning a primary model, we created a secondary model to compete in a bonus competition where students needed to flag images not represented by the character set in ϕ. The class of images not represented by ϕ is called γ, where Ω is the set $\{\phi, \gamma\}$ such that $P(\Omega) = 1$. Ω is called the *hard* dataset used to evaluate models in the bonus competition (Section III-C).

II. Implementation

This section discusses data collection and preprocessing for the easy test set ϕ (Section II-A) and for the hard test set Ω (Section II-B).

A. Implementation for Easy Dataset

The ϕ dataset was collected from 67 students, where each student hand-wrote and photographed 10 samples for each character in the character set β = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', '$', '#'}. Each student labeled their images, resulting in a total of more than 6700 images. Class labels for each character were manually verified multiple times.

Image pre-processing included image erosion, applying the morphological gradient, image resizing, and normalization. These preprocessed images were used in several experiments to identify a model that could classify the character set with high test accuracy. These experiments (Experiments 1 and 2 in Section III) surveyed an array of model types and down-selected the CNN model type for hyperparameter tuning.

B. Implementation for Hard Dataset

Theoretically, the γ dataset covers a larger feature space than is represented by ϕ. However, we let the cardinality of γ and ϕ be nearly equal ($|\gamma| \approx |\phi|$) in an attempt to prevent model overfitting. Dataset γ was generated with the program Imageye [6] which was used to download random images from the Google search engine for the following categories: animals, people, ceremonies, household objects, vehicles, outdoor scenery, and computer-generated characters and digits not represented by classes in ϕ. The authors' *a priori* beliefs around the underlying distribution of Ω placed slight emphasis on images resembling dogs given intuition of Professor Silva's affinity for the domesticated wolf descendant. Aside from this exception, the number of samples in each

category was randomized. Note $|\gamma|$ = 6,871, which is 2.25% greater than that for ϕ.

Images collected for γ underwent the same preprocessing steps outlined for ϕ (II-A). The resulting Ω was used in a single experiment (Experiment 3 in Section III) to train a filter neural network (FNN) that discriminates between ϕ and γ. Further details are presented in Section III-C.

III. Experiments

Three experiments were performed to identify a model that could classify hand-written characters. Experiment 1 evaluated the test accuracy of several low-complexity model types (Section III-A). CNNs performed best from the initial model types, so Experiment 2 iteratively enhanced the CNN model architecture and tuned the learning parameters to minimize the testing error and achieved a 95% test accuracy for the easy data set (Section III-B). Experiment 3 extended the model architecture to filter images belonging to the γ image set using a secondary FNN (Section III-C). The FNN had a 98% detection rate for the γ test samples (See Section II-B). For all experiments, 30% of the training data was used as validation data during K-fold cross validation.

A. Experiment 1 to Down-Select a Model Types

Six model types were selected to evaluate performance when trained using the ϕ dataset. The model types included NB, *k*-NN, LDA, LGA, SVMs, and CNNs. Table I summarizes the parameters used to train each each model in Experiment 1. The image resolution was 75x75 pixels for all models.

Only two model types performed close to or above 50% accuracy for the first iteration, including the SVM and CNN classifiers. Note that SVM used the radial basis function (RBF) kernel, which is known to be powerful in finding decision boundaries in high dimensions. The CNN model is different than the other models as it encodes spatial information into the model architecture. This information likely improves the test accuracy because images contain many spatially-relevant patterns. Because the CNN worked best for this dataset, this model type was selected for hyperparameter optimization in Experiment 2.

B. Experiment 2 to Enhance CNN Architecture

The general CNN architecture is displayed in Fig. 1 showing an input layer (Fig. 1.1), convolutional layers (Fig. 1.2), fully connected layers (Fig. 1.3), and an output layer (Fig. 1.4). In addition to the learning algorithm hyperparameters, each layer type corresponds to a set of variables that needed to be optimized:

1) The input layer can vary in image resolution. Tests indicated that an image size around 150x150 pixels performed best. This parameter was held constant for Experiments 2 and 3.
2) CNN layers can vary in number of layers, kernel size, pooling size, and application of batch normalization and dropout.

TABLE I

Summary parameters used when training six models in Experiment 1. Many parameters represent the defaults present in the scikit-learn python library [7]. The starting CNN parameters are from software given by Haotian Yue [8]. The test accuracy of each model type is given beside the model name. Layer # is abbreviated as (L#). Fully Connected is abbreviated (FC).

Naive Bayes (NB)	Test Accuracy: 21.3%
Class Count	10
Variable Smoothing	1e-9
K-Nearest Neighbors (*k*-NN)	Test Accuracy: 21.1%
K neighbors	4 (Best within 1>=K>=10)
Weighting	Uniform
Distance Metric	Minkowski
Leaf Size	30
Linear Discriminant Analysis (LDA)	Test Accuracy: 19.9%
Components	9
Solver	Singular Value Decomposition
Shrinkage	None
Logistic Regression (LGA)	Test Accuracy: 27.8%
Solver	Limited-memory BFGS
Penalty	L2
Tolerance	1e-4
Support Vector Machine (SVM)	Test Accuracy: 49.1%
Kernel	Radial Basis Function
Regularization parameter (C)	10
Convolutional Neural Network (CNN)	Test Accuracy: 73.9%
Batch Size	64
Epochs	25
CNN-FC Layers	2-2
CNN Filter Size	32 (L1); 64 (L2)
CNN Kernel Size	5 (L1); 5 (L2)
CNN Pooling Size	2 (L1); 2 (L2)
CNN Batch Normalization	None
FC Layer Size	1024 (L1); 10 (L2)
FC Layer Activation Function	Relu (L1); Softmax (L2)

3) Fully Connected (FC) layers can vary in number of layers, number of nodes per layer, and application of batch normalization and dropout.

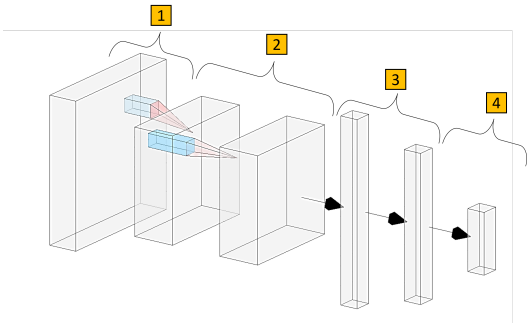

Fig. 1. Template CNN architecture with (1) an input layer, (2) a set of convolutional layers, (3) a set of FC layers, and (4) the output layer.

Experiment 2 manually performed alternating optimization where one parameter was changed at a time to reduce the test error. Table II sequences parameter changes that improved model accuracy. This table can be interpreted to understand potential benefits in tuning each parameter.

Increasing the number of convolutional layers from two to five increased test performance by nearly 4%, likely because the model could find more compressed objects in the feature

space. Past five layers, model performance did not improve significantly.

Increasing the number of FC layers improved the model performance by nearly 3%, likely because more degrees of freedom are available to map input images to the output label. Excessive layers significantly increased the training time, so only five were used.

Changing the CNN kernel size per layer decreased test accuracy by nearly 3%, so the kernel size was kept constant at 5 for the rest of the experiment. It is unknown whether further optimization of this variable could improve performance.

Using a series of reduced batch sizes increased the test accuracy by nearly 8%, likely because decreasing the batch size increases the rate of convergence to local minima. However, the variability of the learning curve typically increases which is why a second set of epochs was used with an increased batch size. This approach had the largest improvement on the CNN performance.

Using batch normalization and dropout after every convolutional layer increased the test accuracy by about 2%. Normalization helps keep feature distances similar between consecutive layers, which helps the backpropagation algorithm converge. Dropout was applied after every convolutional and fully connected layer. The probability of dropout was low (20%) in the first two convolutional layers because these two layers are responsible for extracting features from the images and we did not want to lose essential information. The probability increased to 30% in the third and fourth convolutional layers because this is the point where features significance is weighed. The probability dropped back down to 20% in the last convolutional layer because this is the point where important features are consolidated. The probability in the fully connected layers remained high (30% to 40%) because this is where features are being pieced together to identify which classes the images belong to. Having a higher dropout probability in the fully connected layers can reduce the effects of noise on classification.

Combining these changes produced the final CNN architecture. Finally, grid search was used to optimize the batch size and number of epochs with respect to the testing loss for batch size in the domain $B : [2, 64]$ and epochs in the domain $E : [5, 15]$. As seen in Fig. 2, the test accuracy achieved approximately 95% after three successive training iterations as specified by ID 7 in Table 2 (batch size: 4, 9, 12; epochs: 6, 9, 12). Furthermore, both the training and test accuracies were very close to each other which usually indicates that the model is not overfitting. Fig. 2 also shows both the training and test loss decrease as training epochs increase, though they stop improving around 24 epochs. The resulting confusion matrix is in Fig. 3.

C. Experiment 3 to Adapt the CNN for the Hard Dataset

Experiment 3 tested a second CNN model trained as a filter (or the FNN), which flags images as the γ class. During hard mode testing, a test image is first passed through the FNN. If the image is classified as belonging to γ, it will automatically

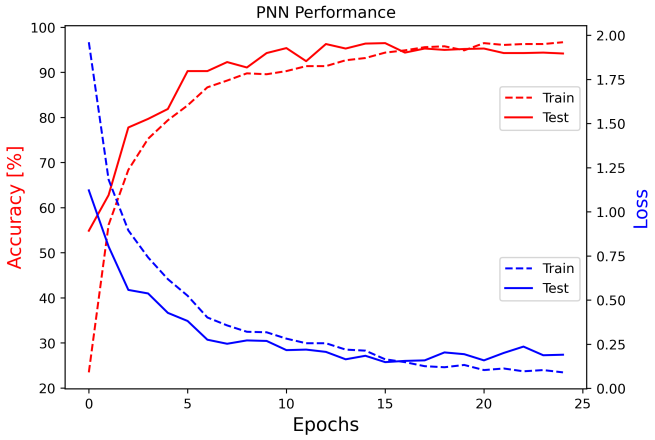

Fig. 2. Training and test accuracy and loss for each epoch when training the primary CNN for ϕ.

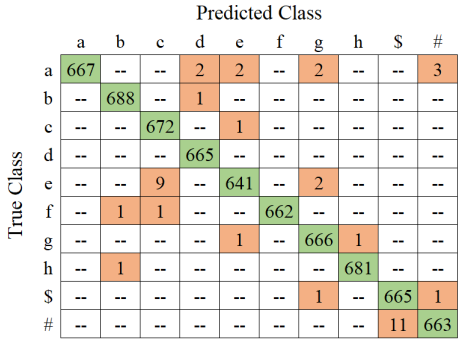

Fig. 3. Multiclass confusion matrix for the trained primary CNN for ϕ.

be labeled as -1. If the image is classified as belonging to ϕ, then the image proceeds to the Primary NN (PNN) for further classification. The PNN is the same model trained in Experiment 2.

The FNN architecture duplicated the PNN, or row 6 from Table II. Both the PNN and the FNN architecture are exactly the same except that the FNN outputs two classes instead of the ten used in the PNN. As the FNN architecture was held constant, Experiment 3 focused on using grid search to find the optimal batch size and number of epochs for the FNN.

When training the FNN, all images within ϕ were labeled as 1 and all images within γ were labeled as 0. Training the learning parameters found that a batch size of 4 for the first 6 epochs and a batch size of 9 for the last 5 epochs produced the highest testing accuracy. Fig. 4 shows the filter accuracy and loss, including the confusion matrix for the FNN.

IV. Conclusions

Salient conclusions from this project are seven-fold: (1) Evaluating a set of model types is important to achieving good performance in classification tasks. (2) The dataset seems to impact the best model selection. For this project, the CNN classifier performed well as may result from encoding the spatial information contained in the handwritten character

TABLE II

Summary of Experiment 2 to enhance CNN model architecture. All models were trained using the Adam optimizer with a learning rate of 1e-4, and used the categorical cross-entropy loss function. The architecture defines a set of convolution layers (C-Layer) which are flattened prior to entering a set of FC layers. All layers used the Relu activation function unless marked as Soft Margin (SM). Experiment 2 was performed using Tensorflow Keras, though the final model was translated to and optimized using the Pytorch library. Cells with semicolon-separated values infer how a variable changes between consecutive convolutional (C) or FC layers. In each row, blue and bolded text illustrates which parameter was changed in a given sub-experiment. The given values per row represent the best of several tested values.

ID (Test Accuracy)	Batch Size (Epochs)	Layers (C-FC)	C-Layer Kernel Size	C-Layer Filters (Act. Func.)	C-Layer Pooling Size	C-Layer Batch Normalization and Dropout	FC Layer Nodes (Act.)
1 - (77.8%)	64 (8)	**C5**-FC2	5;5;5;5;5	32;64;**64;64;64**	2;2;**2;2;2**	None	1024;10(SM)
2 - (81.6%)	64 (9)	C5-**FC5**	5;5;5;5;5	32;64;64;64;64	2;2;2;2;2	None	1024;**512;256;64;**10(SM)
3 - (77.9%)	64 (7)	C5-FC5	**20;16;8;4;2**	32;64;64;64;64	2;2;2;2;2	None	1024;512;256;64;10(SM)
4 - (82.6%)	64 (15)	C5-FC5	5;5;5;5;5	32;64;64;64;64	**4;3;3;2;2**	None	1024;512;256;64;10(SM)
5 - (90.3%)	**4;8 (10;2)**	C5-FC5	5;5;5;5;5	32;64;64;64;64	4;3;3;2;2	None	1024;512;256;64;10(SM)
6 - (92.0%)	4;8 (10;2)	C5-FC5	5;5;5;5;5	32;64;64;64;64	4;3;3;2;2	**Used**	1024;512;256;64;10(SM)
7 - (95.0%)	**4;9;12 (6;9;10)**	C5-FC5	5;5;5;5;5	32;64;64;64;64	4;3;3;2;2	Used	1024;512;256;64;10(SM)

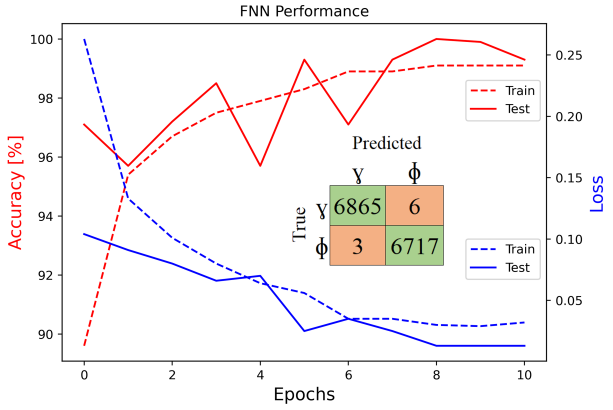

Fig. 4. Training and test accuracy and loss for each epoch when training the filter CNN for Ω. The two class confusion matrix is included in the figure. A batch size of 4 was used for 6 epochs, followed by 5 epochs with a batch size of 9. This change reduces the variability in the learning learning curve.

images. (3) Reducing an image's resolution fights the curse of dimensionality, though the loss of information eventually decreases training performance. (4) There can be a lot of guesswork in finding the appropriate CNNs architecture. Many interdependent CNN hyperparameters comprise a large design space which can be hard to explore. Proper experimental design appears important to achieving high model performance. (5) The batch size can significantly impact the learning algorithm's ability to converge to a high-performing model. (6) Dropout can improve testing performance, but the parameters must be carefully selected to avoid losing important information in the images. (7) Batch normalization can help prevent certain weights from dominating during backpropagation.

V. Acknowledgements

A special thanks to Dr. Silva for teaching the Fundamentals of Machine Learning course and to Haotian Yue for his assistance as a TA this semester.

VI. References

[1] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "EMNIST: Extending mnist to handwritten letters," *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017.

[2] B. Widrow and M. E. Hoff, "Adaptive switching circuits," *Defense Technical Information Center*, 1960.

[3] F. Rosenblatt, "The Perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[4] R. E. Uhrig, "Introduction to artificial neural networks," *Proceedings of IECON '95 - 21st Annual Conference on IEEE Industrial Electronics*, 1995.

[5] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," *2017 International Conference on Engineering and Technology (ICET)*, 2017.

[6] Imageye. "Image downloader - Imageye". In: (2022).

[7] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: Journal of Machine Learning Research 12 (2011), pp. 2825–2830.

[8] Haotian Yue. "cnnPytorch.py". In: (2022).