Final Project Report for CS 175, Spring 2018

Project Title: Handwritten English Alphabet Recognition

Project Number: Team 1

Student Name(s)
Winston Lan, 45942081, wclan@uci.edu
YeeJay Ng, 31741119, yeejayn@uci.edu
Ryan Aveo, 19155229, raveo@uci.edu

1. Introduction and Problem Statement (1 or 2 paragraphs)

Our project is similar to the classic machine learning problem of classifying handwritten digits, but instead our project's goal is to classify handwritten english alphabet characters. For example, given a hand written letter of the vowel 'a', the model will be able to identify the handwritten character as the vowel 'a' and output 'a'. We will use Convolutional Neural Network methods to predict the correct letter using the Handwritten A-Z data set. Our best model, an ensemble model of three different CNN architectures, was able to achieve a test accuracy of 0.9925 after training with 20 epochs.

2. Related Work: (1 or 2 paragraphs)

In the past, the application of a convolutional neural network has been successful in producing results over 90% on MNIST datasets of handwritten digits [1]. Namely the LeNet 1 Model developed by researchers at Bell Labs was able to achieve a classification error rate of 1.7% on the MNIST data set using a multilayered convolutional neural network [1]. A previous neural network model with 2 hidden layers classified 25 x 25 english alphabet images with 82.5% accuracy [2].

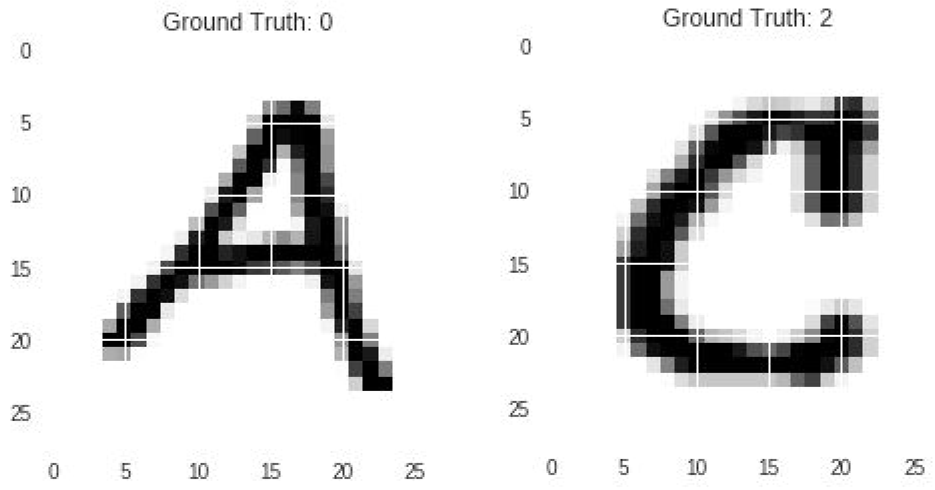
Our project hopes to take these previous works a step further by classifying 26 different english alphabet letters instead of 10 digits using convolutional neural networks. We wrote a LeNet 1 model to establish a baseline for our project to see how the original model that was developed for 10 digit classification would perform on the handwritten english alphabet character classification. In addition to using the results we got with our baseline model, we wrote three different convolutional neural network architectures that use hyperparameters and methods that generally work well for classification. For example, we used categorical cross entropy as our loss function to train our models which is the standard loss function for classification problems,.

References

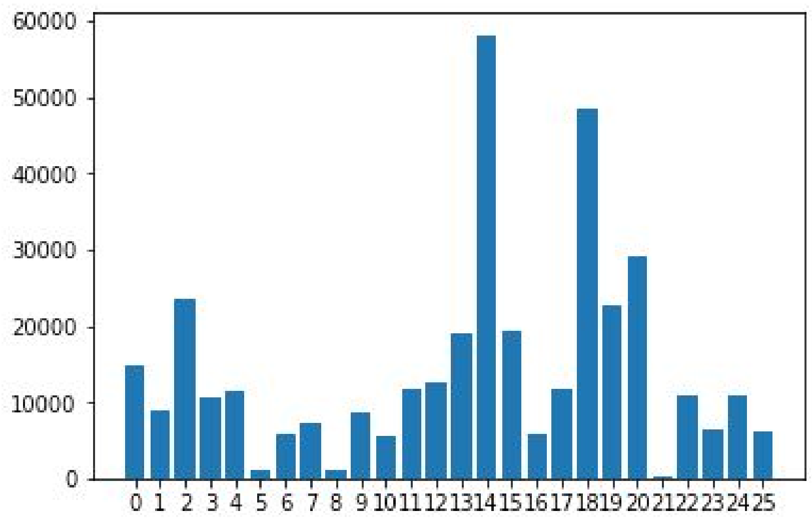
1. LeCun, Yann et al. "Learning Algorithms for Classification: a Comparison on Handwritten Digit Recognition." (1995).
2. Perwej, Yusuf, and Ashish Chaturvedi. "Neural networks for handwritten English alphabet recognition." *arXiv preprint arXiv:1205.3966* (2012).

3. Data Sets

We are using an MNIST dataset of handwritten alphabet images. There are 372,037 images with a dimension of 28x28 with 784 features, and one classification label. There are 26 different classification labels, corresponding to the 26 different letters in the english alphabet. Here are some example images in the dataset:



The distribution of the data was not uniform as some characters had more data samples than others in the overall dataset, as seen in the following bar graph:



The data contained in the matrix had a range of [0, 255] to represent a grayscale image, very much like the original MNIST dataset.

Kaggle Dataset: https://www.kaggle.com/ashishguptajiit/handwritten-az/data

4. Description of Technical Approach [at least 1 page]

To solve our image classification problem we decided to implement three different model architectures and compare their results with one another. To establish a baseline for our own models, we implemented a LeNet 1 Model. Because all three different model architectures did extremely well on test data, there wasn't a need for hyperparameter tuning, so we decided to combine the three models in an ensemble model that took the average of the three. Doing so actually increased our accuracy to around 0.99.

To preprocess the data, we normalized and standardized our data set in order to reduce the range of values for our models to train on. This reduces the range of each value in the 28 x 28 image representation from [0, 255] to [-1, 1]. We subtracted the mean of the training data from the training and test data, and divided them by 255. Standardizing and normalizing the data helps the model train efficiently since it doesn't need to perform large arithmetic calculations because of the reduced values.

To build our models we used the Keras API sequential model. Initially, we used Tensorflow to develop our models but found that we could quicken the process by using Keras' high-level API. In addition, we worked on trying different weight and bias initializations but this didn't change our accuracies significantly. Because of this, we used the Keras default initialization which is a Glorot Uniform Initializer.

The first model we developed was the LeNet 1 Model developed by Yann LeCun to classify handwritten digits. The purpose of using the LeNet 1 Model was to establish a baseline performance for our own model architectures as well as potentially provide us any ideas in developing our own model architectures. After developing the LeNet 1 model, we found that it performed horribly on our dataset. We decided to modify the original LeNet 1 model and added additional layers and filters which improved the LeNet 1 model's performance. This showed us that we needed more filters and layers in our own models.

The three model architectures that we wanted to implement for our image classification project were:

- [conv-relu-pool]xN --> [affine]xM --> [softmax or SVM]
- [conv-relu-conv-relu-pool]xN --> [affine]xM --> [softmax or SVM]
- [batchnorm-relu-conv]xN --> [affine]xM --> [softmax or SVM]

These three model architectures were suggested to us through the homework assignments that we completed during the course. For our project we decided to implement each model, train each model, test each model, and compare the results with one another to evaluate model performance. Each of these models utilize an iteration of convolutional network layers, into multiple affine layers, and finally a softmax or SVM activation function. Each specific model is detailed in our individual contribution papers.

To evaluate our models, we experimented with using both cross-validation and with reserving 20 percent of our training data as validation data. We wrote a script that used sklearn's "model_evaluation" module to perform five fold stratified cross-validation on our models; however, we found that this was computationally expensive compared to just reserving a part of our training data as validation data, so we used the latter method for evaluation since both methods evaluated our models with similar accuracy.

5. Software [at least ½ a page]

Code/Scripts written by us

- Data Preprocessing Script
    - Input for the data preprocessing script consists of the 372,037 csv file that stores the handwritten A-Z data set.
    - Output for the data preprocessing script produces a N x 784 matrix for the X values, and N x 1 matrix for the Y values/labels.
    - Script standardized and normalized data set by subtracting the mean and dividing by 255.
- LeNet 1 Model
    - LeNet 1 Model takes an input of a data set of 28 x 28 images, and outputs the classification results for each image in the data set.
    - Based on the LeNet 1 Model developed by LeCun for handwritten 10 digit classification.
- LeNet 1 Model Modification
    - LeNet 1 Modification Model takes an input of a data set of 28 x 28 images, and outputs the classification results for each image in the data set.
- Convolutional, Relu, Convolutional, Relu, Pool Neural Network Model
    - This model takes an input of a data set of 28 x 28 images, and outputs the classification results for each image in the data set.
- Batchnorm, Relu, Convolutional Neural Network Model
    - Batchnorm, Relu, Convolutional Neural Network Model Model takes an input of a data set of 28 x 28 images, and outputs the classification results for each image in the data set.
- Convolutional, Relu, Pool Neural Network Model
    - Sequential Keras Model with the above layers.
    - Optimized with Adam with a learning rate of 1e-3.
    - Loss calculated by categorical_crossentropy.

Code/Script Written by Others

- Keras API
    - Implement the Convolutional Neural Networks.
    - Test and train our models.
- Numpy
    - Shuffle data.
    - Compute matrix arithmetic.
- Matplotlib
    - Plot accuracy and loss on training and test data.
    - Display example images from dataset.
- Sklearn

    - Model evaluation using cross-validation.
    - Split data into training, validation, and test data.
- Google Colab
    - Accelerate training time with GPU.

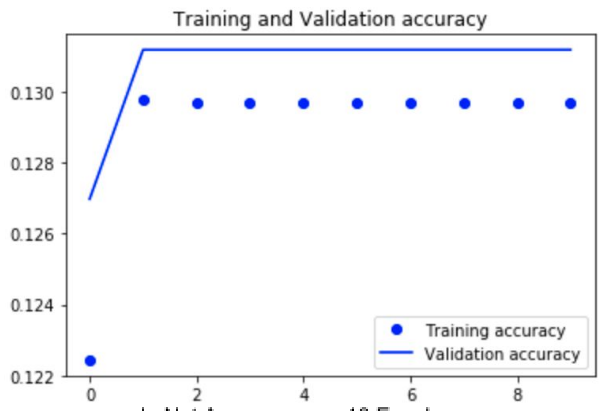
6. Experiments and Evaluation [at least 1 page]

First, we worked on our own separate model architectures to find the most effective CNN architecture. We used categorical cross entropy to calculate our loss which is generally used for classification problems. We wanted to tune the hyperparameters of the architecture with the highest accuracy; however, all of our models performed equally as well. Because of this, we made an ensemble of all of our models instead of tuning hyperparameters.

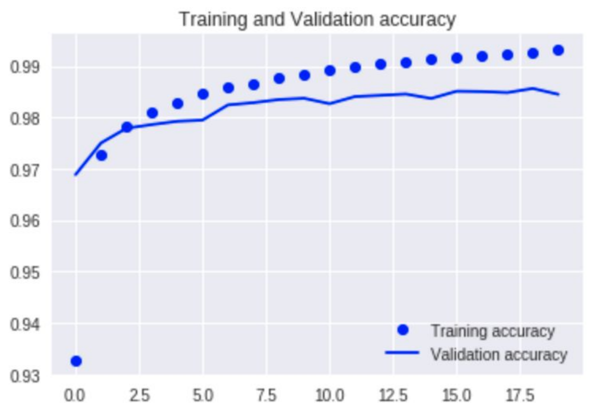
During experimentation, we added a variety of different layers to the models to extract key features of our images. While increasing the number of filters and changing the filter size to a smaller value increased accuracy, we noticed that adding many more layers usually increased overfitting.

Because our project is a classification problem, we evaluated our accuracies based on the percentage of correct classifications. Each of our models would output an array of predictions for each image, which we could then compare to the true classes of our validation and test sets. As mentioned above, we evaluated our models using cross-validation during our initial phases of testing, but decided to not continue using it due to the expensive computational cost. The resulting testing accuracy after cross-validation did not differ widely, if at all, from evaluating our model with reserved validation data partitioned from the test data.

During testing, we plotted graphs of our accuracy and loss to visualize the effectiveness of our changes and how those changes affect the model over a number of Epochs. Below is a graph of our base model's (LeNet 1's) accuracy:

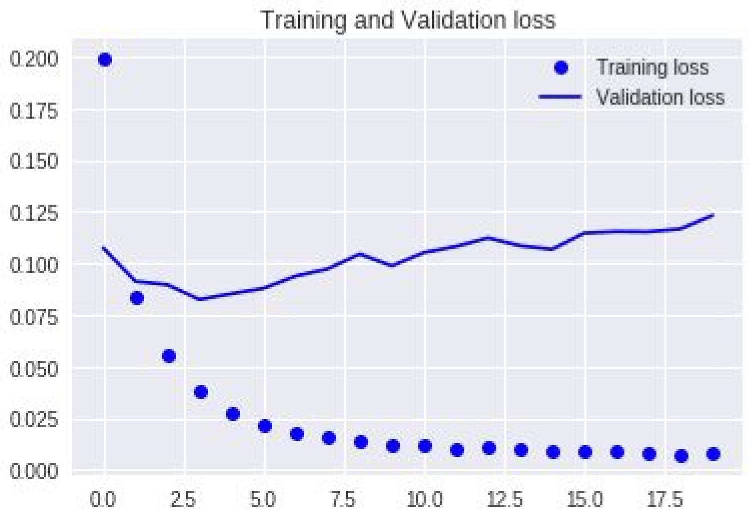


LeNet Accuracy over 10 Epochs



Convolutional, Relu, Convolutional, Relu, Pool over 20 Epochs

Visualizing the training and validation accuracy against our epochs made it easier to detect overfitting. For example, sometimes the graph would show an excellent accuracy on training data but a significantly worse accuracy on the validation data. This would be supported by the loss graph as certain periods between epochs showed an increase in loss rate rather than a steady decrease. If the model was not overfitting, then the validation loss would continue to decrease, inverse to the validation accuracy increasing. Below is a graph of the loss of one of the model's we developed (Batchnorm, Relu, Convolutional) that demonstrates the gradual overfitting of our models.



Batchnorm Relu Convolutional Model Loss Over 20 Epochs

Despite the overfitting our models were still able to perform extremely well. The performance of all the models we tested after 20 epochs of training is as follows:

Model	Test Performance
LeNet 1	Accuracy: 0.0678 Loss: 9.7016
LeNet 1 Modification:	Accuracy: 0.9911 Loss: 0.0571
[conv-relu-pool]xN --> [affine]xM -> [softmax or SVM]	Accuracy: 0.9885 Loss: 0.0667
[conv-relu-conv-relu-pool]xN -> [affine]xM -> [softmax or SVM]	Accuracy: 0.9842 Loss: 0.0726
[batchnorm-relu-conv]xN -> [affine]xM -> [softmax or SVM]	Accuracy: 0.9854 Loss: 0.1155
Ensemble:	Accuracy: 0.9925

	Loss: n/a

Given the high performance of all of our models, we decided to combine all three of our models into an ensemble. Our ensemble classifier had a successful classification rate of 0.9925. This was most likely due to multiple classifiers positively reinforcing one another about their classification of an image. Each of our models outputs an (N,26) array of class probabilities for each image. The corresponding letter with the highest probability would then be our classifier's prediction. Our ensemble takes the outputs from each model and averages them, allowing for errors in one model to likely be corrected by the correct predictions from the other two. Because our models all performed equally well, we chose to evenly weight the decisions of each model in our ensemble. Had a model been less successful than the others, we would have chosen to give it less of an impact in the final output.

7. Discussion and Conclusion [at least ½ a page]

One major insight we were able to gain from this project was the powerful ability of convolutional neural networks in image classification problems. With just a simple model consisting of one or two convolutional neural network layers and just a few iterations, the machine learning algorithm was able to already correctly classify the handwritten english letters with over 95% accuracy.

A surprising outcome was the original LeNet 1 model was only able to achieve under 20% accuracy which was a surprise because it was able to correctly classify handwritten digits with over 90% accuracy. This showed that even though our project is a similar problem to digit classification, we needed to tune hyperparameters such as filter size and number for our problem of classifying handwritten english characters.

In the end, the high test accuracy came as a surprise to us because we expected a more complex model would be required in order to achieve a high accuracy. Additionally, we expected that more training iterations would be needed to train our model to be as successful as it currently is. However, just after 5 iterations, all of our models reached over 98% accuracy. The ensemble classifier performed as we expected, achieving over 99% classification accuracy.

In the future, we would like to see even more progress on this problem by creating models to classify handwritten words and even whole documents. With such a high accuracy that our models achieved on just singular handwritten characters, we believe that expanding this problem to whole handwritten documents is possible.