M508 Group Project
Handwritten Character Classification Using Convolutional Neural Net

Chris Corona cjcorona11@gmail.com
Nolan Walker nolantwalker@gmail.com

1 Introduction

Computer vision is an emerging field where the goal is to train computer systems to understand and interpret the visual world by extracting meaningful information from digital images and video. Applications include face identification, medical imaging classification, object recognition, and self-driving cars. The main technology behind computer vision is the convolutional neural network (CNN). This is a type of deep learning that has evolved specifically for image data. In this project, we apply a CNN to classify handwritten characters in tiny images from a large dataset. Our experimentation includes several instances of the CNN which cover some very basic hyperparameter tuning. In the end, this deep learning model achieves modestly high classification accuracy on the validation set.

2 Dataset

MNIST is a widely known dataset of handwritten digits (0-9) that was originally used by Yann LeCun in their 1998 paper on document recognition. It quickly became a "toy" dataset because of the ease of obtaining high accuracy with various models. As a result, Gregory Cohen published a more challenging version known as the extended MNIST (EMNIST) dataset in 2017. This dataset comes from the same NIST source except that it also includes letters in addition to all the digits. So there were 47 classes: digits 0-9, uppercase letters A-Z, and lowercase letters a, b, d, e, f, g, h, n, q, r, t. They omitted any lowercase letters that closely resembled their uppercase counterparts (or likely just grouped them into their respective uppercase class). We chose to use Cohen's EMNIST Balanced dataset because it had a reasonable size of 131,600 images with balanced classes. This way it wouldn't require excessive computing power to train, and we wouldn't have to deal with issues of imbalance. Each image is 28 x 28 pixels in grayscale and is fully pre-processed, meaning they are size-normalized, centered, and fixed-size. This allowed for immediate deployment in the machine learning model; no data wrangling necessary. The details of the dataset are shown in Figure 1 and a sample of the dataset is shown in Figure 2. Cohen's purpose was to release a more challenging dataset than the original MNIST which had become almost trivial with modern techniques. Figure 3 shows the accuracy of neural nets with various numbers of hidden nodes on the EMNIST and original MNIST datasets (using the exact same network). In the following sections, we demonstrate how much more challenging the EMNIST dataset actually is.

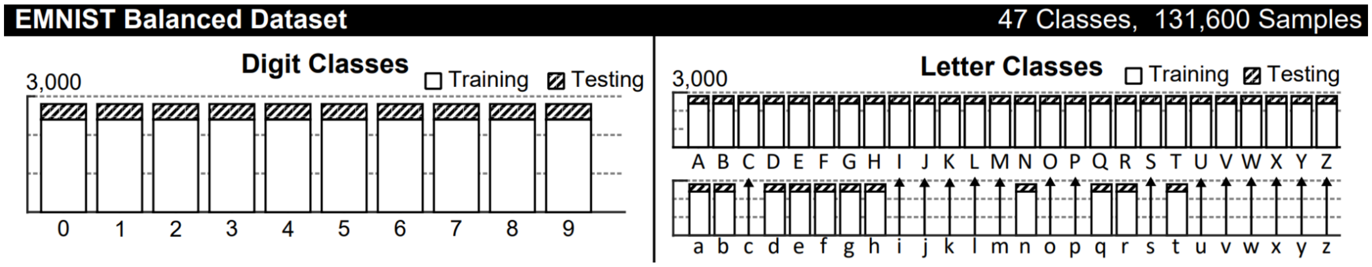


Figure 1: Details of the EMNIST dataset.

Figure 2: A sample of the EMNIST dataset.

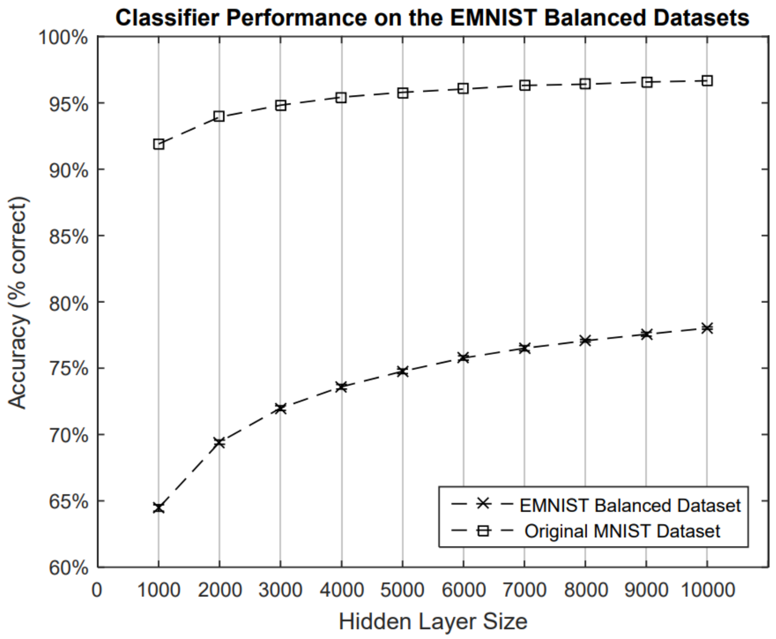


Figure 3: Performance of Cohen's neural nets on the EMNIST and original MNIST datasets. Note that the exact same networks were used on both datasets for a fair comparison.

3 Methods

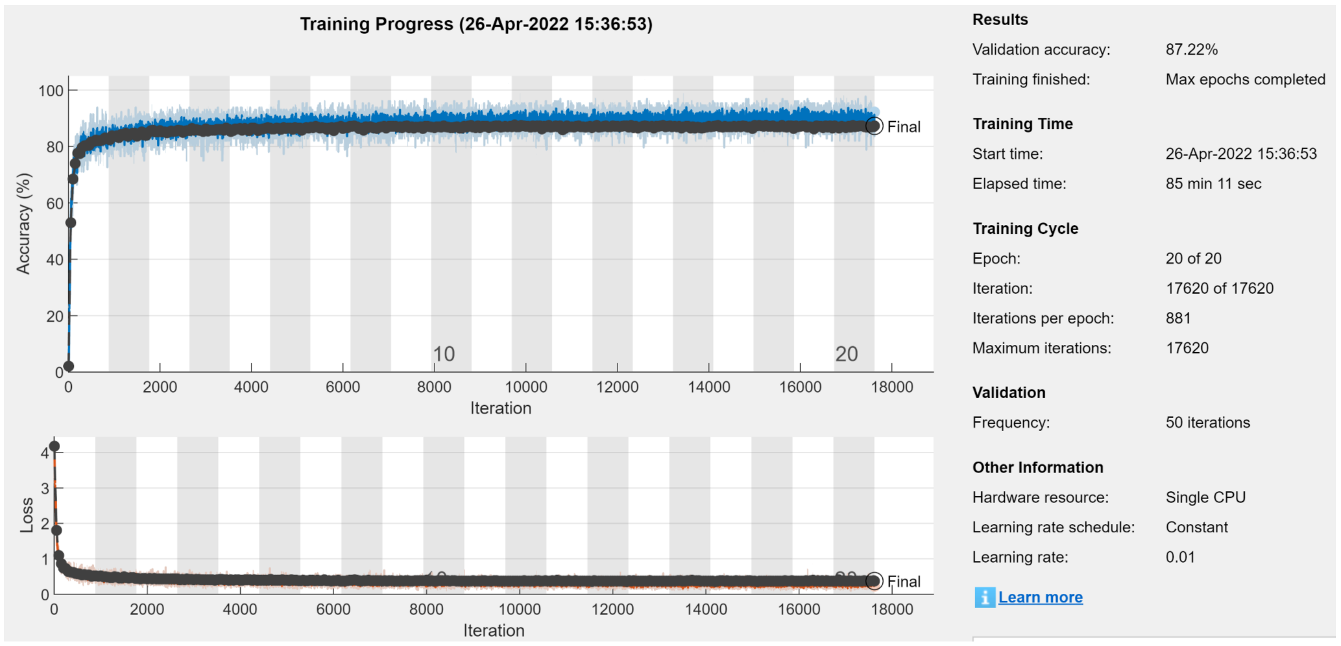
Convolutional neural nets represent the state-of-the-art for various image classification problems. It was an obvious choice for us to apply a CNN to this problem. But in addition to that, we also had a personal desire to learn more about them and to try to train one on a real machine learning task. So let's dive into the basics of CNNs. They are composed of convolutional layers, pooling layers, and a final fully-connected layer at the output. A convolutional layer takes a filter or window of a specified size and passes it across the two-dimensional (or 3D for color) image input. The filter combines the pixels within the window in a specified way - usually a dot product (eg. multiplying pixels along the diagonal by 1 and those on the off-diagonal by 0, then summing). The filter can then slide along one pixel at a time or it can jump by more pixels. This is known as stride. Padding is used to make sure the filter window gets proper coverage of the edge pixels. The number of filters is also specified for each layer, with more filters mapping more features at its output. The filters, stride, and padding components make up the convolutional layer. Next is the pooling layer which has the purpose of reducing dimensionality. Here the filter window passes over the two-dimensional (or 3D for color images) output from the convolutional layer. The two main types of pooling are max and average. Max pooling is the most common, and it simply takes the largest value from the window and sends it to the output. Similarly, average pooling takes the average from the window. Several convolutional and max pooling layers are stacked one after the other to reduce dimensionality down further and further. In theory, the convolutional layers will have extracted high-level features from the original image data. This reduced image data is input to the fully-connected layer. This is just a regular neural network whose job is classification via a final softmax layer.

4 Experiments

The dataset came with predefined splits of 112,800 training samples and 18,800 testing samples. We trained four different CNNs on the dataset. The baseline CNN had three layers of 8, 16, and 32 nodes respectively with a ReLu activation function and 2x2 max pooling layer between each. For each layer, we used the same convolutional filter size of 3x3, batch normalization, "same" padding, and stride of two. See the appendix for more details. All of the nets had the same training options: stochastic gradient descent method with 20 max epochs and shuffled the data between each epoch (again, the specifics are in the appendix). We trained other networks by changing only one piece of the architecture from the baseline as follows. We tried training CNNs with more layers of greater node size (64,128) and with the same amount of nodes. We also tried ordering the layers by decreasing the number of nodes per layer. Since our network is quite shallow, we also tried changing the activation function to a sigmoid because ReLu improves the performance of deep networks. And finally, we tried removing the first max pooling layer. The reasoning for this comes from the small image sizes in the dataset; we don't need to perform aggressive dimensionality reduction with so many (and early) max pooling layers.

5 Results

The baseline convolutional neural network with three layers of 8,16, and 32 nodes respectively, max pooling layers in between convolution layers and Relu activation functions trained on the original MNIST dataset learned to classify the data with very high accuracy (~99%). The same CNN trained on the EMNIST dataset was able to obtain accuracy of around 87% as shown in Figure 4. This is a modest improvement on Cohen's 78% accuracy on the EMNIST dataset. Modifying the CNN architecture didn't substantially increase the accuracy of any trained nets - the baseline had the best results.



Results

Validation accuracy:	87.22%
Training finished:	Max epochs completed

Training Time

Start time:	26-Apr-2022 15:36:53
Elapsed time:	85 min 11 sec

Training Cycle

Epoch:	20 of 20
Iteration:	17620 of 17620
Iterations per epoch:	881
Maximum iterations:	17620

Validation

Frequency:	50 iterations

Other Information

Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	0.01

Figure 4: Results of our best CNN trained on the EMNIST dataset.

6 Discussion

The difference in accuracy of the trained CNNs on the two datasets is most likely due to the similarity between some characters in the EMNIST dataset. Whereas the numbers are quite distinct, some of the letters of the alphabet were probably not distinct enough from other letters or numbers to be classified correctly with a high probability(e.g. g and 9, etc.). These are drawbacks of the dataset, as there may be some characters that are not separable. This makes sense as handwritten writing can be hard to read sometimes, and the positions of characters within a word or sentence clue us in as to what it is supposed to be. It would be interesting to see if any of the more complex pre-trained deep neural nets available to the public could achieve better accuracy on the EMNIST dataset.

7 Conclusion

We showed that Cohen's extended MNIST dataset does offer a more challenging classification problem than the original. We were also able to get better accuracy than Cohen's instructional baseline. The pre-processing of this EMNIST dataset allows for immediate plug-and-play to focus entirely on CNN training instead of time spent on data wrangling. As a result, this is a fantastic dataset for an introductory classification problem with CNNs.

8 Future Work

There are a multitude of different CNN architectures that could be applied in future work. There is much room to improve on our best 87% accuracy. We trained on the most straightforward dataset offered by EMNIST; there were larger and unbalanced versions that would make for even more challenging problems.

9 References

Cohen, Gregory, et al. "EMNIST: An Extension of MNIST to Handwritten Letters". *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017.

LeCun, Yann, et al. “Gradient-Based Learning Applied to Document Recognition”. *Proceedings of the IEEE*, 86(11): 2278-2324, November, 1998

LeCun, Yann. "The MNIST database of handwritten digits." *http://yann.lecun.com/exdb/mnist/*, 1998.

10 Appendix

```
%% Handwritten Character Classification with Neural Nets
% M 508
% Prof. Dominique Zosso
% Chris Corona and Nolan Walker
% 5/6/22

%% Load and Split Dataset
% load the dataset
% (source: https://www.nist.gov/itl/products-and-services/emnist-dataset)
TrainX = modifiedloadMNISTImages("emnist-balanced-train-images-idx3-ubyte");
% we modified the above function taken from
https://github.com/davidstutz/matlab-mnist-two-layer-perceptron/blob/master/loa
dMNISTImages.m
TrainL = loadMNISTLabels("emnist-balanced-train-labels-idx1-ubyte");
% we obtained the above function from
https://github.com/davidstutz/matlab-mnist-two-layer-perceptron/blob/master/loa
dMNISTLabels.m
TestX = modifiedloadMNISTImages("emnist-balanced-test-images-idx3-ubyte");
TestL = loadMNISTLabels("emnist-balanced-test-labels-idx1-ubyte");

%% Network Layers (for our best CNN)
layers = [
   imageInputLayer([28,28,1])

   convolution2dLayer(3,8,'Padding','same')
   batchNormalizationLayer
   reluLayer
   maxPooling2dLayer(2,"Stride",2)
   batchNormalizationLayer
   convolution2dLayer(3,16,'Padding','same')
   reluLayer

   maxPooling2dLayer(2,"Stride",2)

   convolution2dLayer(3,32,'Padding','same')
   batchNormalizationLayer
   reluLayer
   fullyConnectedLayer(47)
   softmaxLayer
   classificationLayer
   ];

%% Network Options
% training options
options = trainingOptions('sgdm', ...
   'MaxEpochs',20,...
   'Shuffle','every-epoch',...%'InitialLearnRate',1e-5,...
   'Verbose',false, ...
   'Plots','training-progress', ...
   'ValidationData', {TestX, categorical(TestL)'} );
%% Net Training
% let MATLAB do the actual training
net = trainNetwork( TrainX, categorical(TrainL)', layers, options );
%% Demo: Load Examples
example_double = zeros(28,28,9);
figure;
for i = 1:47
   switch i
       case{1}
           example = imread('a1.jpg');
       case{2}
           example = imread('a2.jpg');
       case{3}
           example = imread('a3.jpg');
       case{4}
           example = imread('a4.jpg');
       case{5}
           example = imread('a5.jpg');
       case{6}
           example = imread('a6.jpg');
       case{7}
           example = imread('a7.jpg');
       case{8}
           example = imread('a8.jpg');
       case{9}
           example = imread('a9.jpg');
       case{10}
           example = imread('a10.jpg');
```

```
case{11}
    example = imread('a11.jpg');
case{12}
    example = imread('a12.jpg');
case{13}
    example = imread('a13.jpg');
case{14}
    example = imread('a14.jpg');
case{15}
    example = imread('a15.jpg');
case{16}
    example = imread('a16.jpg');
case{17}
    example = imread('a17.jpg');
case{18}
    example = imread('a18.jpg');
case{19}
    example = imread('a19.jpg');
case{20}
    example = imread('a20.jpg');
case{21}
    example = imread('a21.jpg');
case{22}
    example = imread('a22.jpg');
case{23}
    example = imread('a23.jpg');
case{24}
    example = imread('a24.jpg');
case{25}
    example = imread('a25.jpg');
case{26}
    example = imread('a26.jpg');
case{27}
    example = imread('a27.jpg');
case{28}
    example = imread('a28.jpg');
case{29}
    example = imread('a29.jpg');
case{30}
    example = imread('a30.jpg');
case{31}
    example = imread('a31.jpg');
case{32}
    example = imread('a32.jpg');
case{33}
    example = imread('a33.jpg');
case{34}
    example = imread('a34.jpg');
case{35}
```

```
            example = imread('a35.jpg');
        case{36}
            example = imread('a36.jpg');
        case{37}
            example = imread('a37.jpg');
        case{38}
            example = imread('a38.jpg');
        case{39}
            example = imread('a39.jpg');
        case{40}
            example = imread('a40.jpg');
        case{41}
            example = imread('a41.jpg');
        case{42}
            example = imread('a42.jpg');
        case{43}
            example = imread('a43.jpg');
        case{44}
            example = imread('a44.jpg');
        case{45}
            example = imread('a45.jpg');
        case{46}
            example = imread('a46.jpg');
        case{47}
            example = imread('a47.jpg');
    end
    example_gray = rgb2gray(example);
    example_double(:,:,i) = im2double(example_gray);
    subplot(7,7,i)
    imshow(example);
end

%% Demo: Predict on Homemade Examples
load('net.mat');
key =
['0';'1';'2';'3';'4';'5';'6';'7';'8';'9';'A';'B';'C';'D';'E';'F';'G';'H';'I';'J
';'K';'L';'M';'N';'O';'P';'Q';'R';'S';'T';'U';'V';'W';'X';'Y';'Z';'a';'b';'d';'
e';'f';'g';'h';'n';'q';'r';'t'];
% make predictions on the live examples
NetpredictionLiveExample = zeros(9,47);
for i = 1:47
    NetpredictionLiveExample(i,:) = net.predict(example_double(:,:,i)');
end
[~, NetLLiveExample] = max(NetpredictionLiveExample,[],2); % label predictions
NetLhuman = key(NetLLiveExample) % human-readable predictions
accuracy = sum(NetLhuman == key)/47
```