Character Classification using Convolutional Neural Networks

Rafe Kruse
Herbert Wertheim College of Engineering
University of Florida
Gainesville, USA
rafekruse@ufl.edu

***Abstract*—Character classification is a very large field relevant to applications such as OCR. Presented with the task of classifying letters and unknowns I propose image cropping and scaling and a Convolution Neural Network model for classification following preprocessing. The preprocessing includes the addition of EMNIST data to represent unknowns, cropping of image data whitespace, and scaling to normalize information for the CNN. The CNN is trained through forward passes of the processed images and back-propagation. The network contains two convolutions and three fully connected layers ending with 9 and 2 neurons for the representative classes of the hard and easy tasks respectively. This method achieves great success on given character classification, however, it struggles with unknown identification.**

I. Introduction

Character classification is reliant upon models that can effectively extract relevant high-level features to determine a correct label. In the past, there have been a variety of algorithms such as kNN's, Radial-Basis functions, and standard feed-forward ANN's applied to this problem [1]. However, more recently Convolutional Neural Networks have begun to dominate image classification tasks [2]. For this reason, I have chosen to use a CNN to approach the problem of character classification given our data for this course.

The relevancy and consistency of training data are paramount to the proper training of a model [6]. Because of this there are many methods of processing and cleaning data for a model. Given the problem presented, the given image data is preprocessed using the whitespace cropping and the nearest neighbor scaling algorithm. This results in borderless images that fill the given space similarly to the EMNIST dataset [4]. The EMNIST dataset is also utilized to include 'unknown' samples of characters not within the range 'a'- 'k'.

II. Implementation

A. Unknown Class Identification

The hard task for this project requires classification of characters into the following 9 classes.

('a', 'b', 'c', 'd', 'h', 'i', 'j', 'k', 'unknown')

The given dataset, however, only includes samples and their respective labels for the first 8 classes 'a-k'. In order to handle the unknown class, I opted to add an additional neuron in the final fully-connected layer of network. This additional neuron functions the same as the 'a-k' output neurons for the 'unknown' class.

B. Preprocessing

The dataset given for training consisted of 6400 binary images that varied in size with the restriction that they did not exceed 54x54. There were 800 samples of each character between 'a' and 'k'. Before receiving the data, most of the images were cropped to have a 1-pixel border and minimal white space, however, there remains a portion of the dataset this does not apply to.

To handle the issue of a lack of data within the 'unknown' class to train upon, a random spread of characters and digits from the EMNIST [4] dataset were added and classified as 'unknown'. The EMNIST dataset contains gradient character images, so to account for this difference, any non-zero pixels within the images were converted to a value of 1.

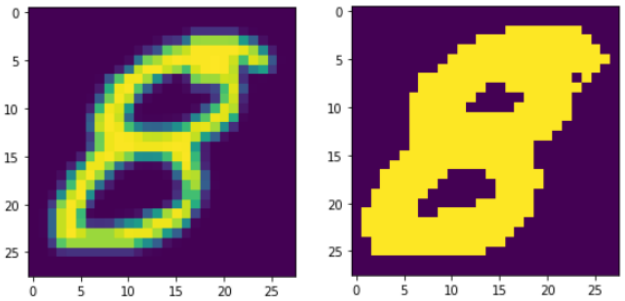

Figure 1) Gradient vs Binary representation of EMNIST digit

With the added EMNIST image there is the issue of the sizes of the dataset of images. All the images must have the same dimensions to be fed into the CNN.

The first processing step is the cropping of the image by removing any row or column containing no non-zero values. This doesn't solve the scale problems; however, it does increase model accuracy and is discussed further in the experiment section below.

The second processing step is to apply the basic nearest neighbor scaling algorithm from the starting size to the desired 54x54 required. This function follows the below steps:

1. *Calculate image dimension ratios*

$$\left.\begin{aligned} x\_ratio &= \frac{w_1}{w_2} \\ y\_ratio &= \frac{h_1}{h_2} \end{aligned}\right\} w_2, h_2 \neq 0$$

2. *For each pixel in the output sample from the input at the same relative position.*

The images results from each of these preprocessing steps can be seen in figure 3, parts a, b, and c.

C. Training

The model created for this classification problem follows the architecture in figure 2 and was implemented using Pytorch. The CNN was trained through iterations of forward passes and backward passes using the standard backpropagation algorithm. Below are the relevant hyperparameters and the respective values used for training of this paper's final model.

- Batch Size: 64

- Epochs: 25
- Learning Rate: 0.001
- Loss Function: Cross-Entropy Loss
- Optimizer: Adam [5]
- Training|Validation data split: 15|1

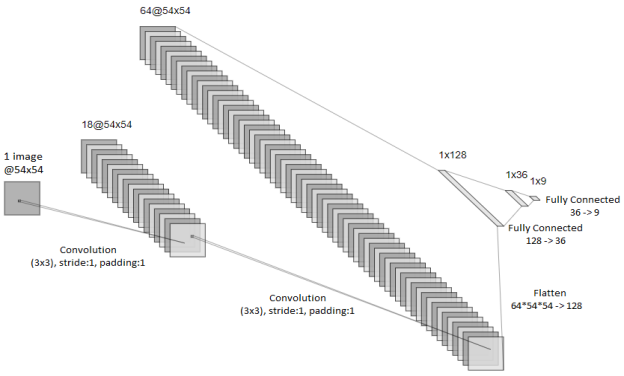

Figure 2) CNN Architecture from Image to Labeling

D. Labeling

The results of the Pytorch model are a vector of confidences that the given data point is a member of each different class. The final label is the max of this vector. Below is an example of the final labeling step.

$$V = [0.532, -0.5932, -15.7, -11.6, 6.5, 3.23, -2.53, 14.6, 4.5]$$

$$O = indexof\big(argmax(V)\big), O = 8$$

$$classes[O] = 'k'$$

III. Experiments

The optimization of the presented model was carried out through general tuning of hyperparameters and experimentation with different implementations. The tuning of the number of training epochs, learning rate, and training validation split, as well as the selection of optimizer and loss function were all found through trial and error testing for the maximum accuracy. Below are three primary experiments that were carried out to better select certain methods for use in the final model implementation.

A. Image Normalization

As previously mentioned in the implementation section the initial size and form of the training data posed a problem for training the model. To simplify testing the smaller 'a'/'b' dataset was used. Using the reduced dataset minimizes the effect of training variance on the accuracy result. This is because it is more likely with a reduced binary dataset that a change in the accuracy will be the result of a change in the observed variable as opposed to a random variation in the CNN's training or initialization. The image normalization testing was on four methods: no-cropping/zero padding, no-cropping/ scaling, cropping/zero padding, and cropping/ scaling.

The goal of this test was to determine the best method for preprocessing as well as determining if the observed outliers in the data would affect classification accuracy. Accuracy results showed that it was ideal for the image to be cropping and scaled as opposed to the usage of zero padding. This result makes sense given that CNN's rely heavily on spatial information. The usage of zero padding resulted in some images resembling Figure 3d while other data that was already the full 54x54 before preprocessing looked like Figure 3c. This posed a problem as they both are the same letter, however, spatially they are very different resulting in different classifications. This cropping and zero-padding implementation achieved a similar result to the size-normalization and centering method used on the EMNIST dataset. Both methods reduce noise relating to letter placement within the image resulting in better feature extraction by the CNN.

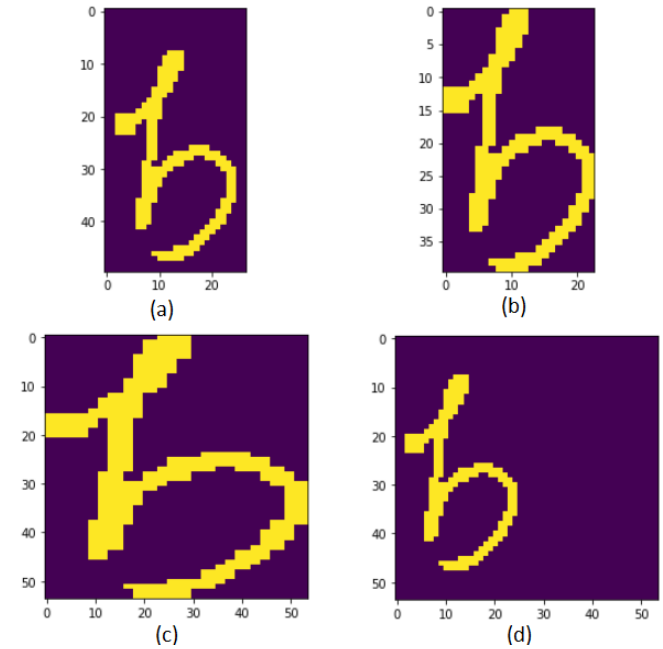

Figure 3) (a) Original image, (b) cropping, (c) cropping and scaling, (d) zero-padding and no cropping

B. Unknown Handling

Two methods were compared for the identification and classification of the unknown class. The first being the addition of a 9th neuron in the output layer of the network and the second being a certainty threshold on the eight existing neurons. Both methods seek to find an effective way of sorting the unknown images from given and trained upon known letters.

The second method relied upon requiring that the argmax value of the network outputs exceeded a threshold otherwise the resulting inputs would be labeled unknown. The reasoning behind this implementation is that ideally, the network will be unsure about the classification of unknown inputs and thus will weight the labeling evenly. This can then be used to categorize non-distinct input images as the unknowns. While this method was successful in identifying unknowns at a high accuracy of 80-90% there was a very high false-positive rate. The model was uncertain about many of the 'a'- 'k' values resulting in a lower overall accuracy than if the unknown values were entirely mislabeled. This is a result of very similar letters such as 'i' and 'j' that end up splitting the model's confidence in the individual values.

The addition of a 9th neuron that represents the 'unknown' class reduces the issue of false positives that occurred in the first method; however, the accuracy of the 'unknown'

classification is much lower and highly variable. This is a result of the fact that the 'unknown' class is unknown, and the only samples of unknown data trained upon were the images taken from EMNIST dataset. However, the unknown could be any image and when random noise or images entirely different from those previously seen in training are presented the CNN is completely lost. This is because CNN's struggle to generalize on data that is completely unique compared to the training data. However, being the more accurate method of classification the 9th neuron was adopted for the final model.

C. *Data Spatial Size Reduction*

It is common practice when working with CNN's to have layers such as max-pooling or larger stride convolutions to reduce model complexity. This allows for quicker training and minimizes lost information.

During my testing I attempted adding a max-pooling or larger stride convolution after every traditional convolution layer. Both implementations with the additional size reduction layers greatly improved the time required for training. Max pooling dropped the time from 10 minutes with 15 epochs to 30 seconds and a larger stride convolution reduced the time to 90 seconds. Both speed-ups are substantial; however, they came with a marginal reduction in the accuracy. Both methods seek to maintain the maximum information possible within the reduced size, however, some minor detail is lost.

It is because of this loss in accuracy that complexity reduction is excluded from the final model. This choice was made based on the restricted dataset that was trained upon and the small number of required epochs to reach model convergence. Given the 6400 'a'-'k' samples and the added 1000 unknowns from the EMNIST dataset. Training time on such as small dataset without model complexity reduction is still minimal. As previously mentioned in the parameter section model accuracy leveled out after 25 epochs and any further iterations were irrelevant. Thus, the purpose of model complexity reduction, minimizing the training time to allow for more epochs and more data, was not useful in the case of this dataset.

IV. Conclusions

From the experiments when working with ANN's it is very clear the importance of having a large amount of data and data that is clean and representative of what the model will be tested upon. The CNN model I used was completely unable to correctly label test data that wasn't normalized or was outside of the range of training information.

During testing and tuning of hyperparameters, I gained a better understanding of the uncertainty that exists within the machine learning field relating to the methodical selection of model parameters. It was clear that the proper parameters for the model were heavily dependent on the training data and very variable from training instance to instance.

It is also clear from my testing on handling unknown data that CNN's have no innate means of identifying unseen information. After working with CNN's this makes sense as the networks attempt to find patterns that are representative of certain classes, which makes finding a pattern in unknown and random images never seen by the model during training impossible.

It is because of this poor 'unknown' classification accuracy that if I could continue to improve, I would look to another model such as a binary 'known'/'unknown' SVM to better identify outliers. I believe the combination of two models one to identify the 'unknown' class and another to classify the given letters would be a more accurate final model.

References

[1] Lee Y. *Handwritten Digit Recognition Using K Nearest-Neighbor, Radial-Basis Function, and Backpropagation Neural Networks*. Mar, 13, 2008. [Online] https://www.mitpressjournals.org/doi/abs/10.1162/neco.1991.3.3.440?journalCode=neco [Accessed Nov. 26, 2019]

[2] Ciresan D., Meier U, Schmidhuber J. *Multi-column Deep Neural Networks for image Classification*. Feb 13, 2012. [Online] https://arxiv.org/pdf/1202.2745.pdf [Accessed Nov. 26, 2019]

[3] LeNail, (2019). NN-SVG: Publication-Ready Neural Network Architecture Schematics. Journal of Open Source Software, 4(33), 747, https://doi.org/10.21105/joss.00747

[4] Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017*). EMNIST: an extension of MNIST to handwritten letters*. Retrieved from http://arxiv.org/abs/1702.05373

[5] Kingma D., Ba J. *Adam: A Method for Stochastic Optimization*. ICLR 2015. Jan 30, 2017. [Online] https://arxiv.org/pdf/1412.6980.pdf [Accessed Nov. 26, 2019]

[6] Rahm E., Do H. *Data Cleaning: Problems and Current Approaches* .Dec 2000 ,Bulletin of the Technical Committee on Data Engineering [Online] https://shorturl.at/swKT3 [Accessed Nov. 26, 2019]