# Convolutional Neural Network and Extreme Gradient Boosting EMNIST Competition

S. H. Magnússon, G. S. Cabrera, and B. S. Dalen

(Dated: Wednesday 18th December, 2019)

Using two machine learning methods, *Extreme Gradient Boosting* (XGB) and *Convolutional Neural Networks* (CNN), this report compares and contrasts their ability to identify handwritten letters and digits from the EMNIST dataset. Their performances are evaluated both with respect to each other, and to the results in *Cohen, G. et al.* "Emnist: an extension of mnist to handwritten letters," where the *OPIUM* classifier was used. Using a grid-search over multiple hyper-parameters, optimized models were created for the XGB and CNN classifiers; it was determined that the CNN yielded the best model with an 86.90% categorical accuracy on a testing set, with XGB at a close second with 82.78% accuracy. Both of these models were able to supersede the effectiveness of the OPIUM model, which had an approximate accuracy of 78.02%.

## I. INTRODUCTION

The application of Machine learning algorithms to the field of image analysis has recently gained exceptional traction within the larger field of artificial intelligence. Image recognition algorithms have been applied to projects such as the up-and-coming self driving car revolution (see *Bojarski, M. et al.* "End to end learning for self-driving cars" [1]) and even in identifying malignant skin cancer (see *Esteva, A. et al.* "Dermatologist-level classification of skin cancer with deep neural networks" [3]).

The project presented will implement two methods on the so-called "Extended MNIST" (EMNIST) data set. The MNIST data set is widely recognized as a good baseline for the evaluation of image recognition algorithms, consisting of a large number of handwritten decimal digits reduced to a 28 × 28 grid of grayscale pixels. The EMNIST data set takes this one step further, and includes handwritten grayscale images of the 52 uppercase and lowercase letters of the Modern English alphabet.

The EMNIST data set has already been studied in great detail – see *Cohen, G. et al.* "Emnist: an extension of mnist to handwritten letters" [2], which the algorithms and methods presented in this research will treat as a benchmark to improve upon.

A *Convolutional Neural Network* (CNN) is a variant of the *Feed-Forward Neural Network*, specifically designed to analyze images and spacial data. In typical Feed-Forward NNs, images are processed by collapsing the pixels into a one-dimensional array: this can be problematic in two ways, especially for high definition pictures.

The first is that high pixel count will lead to large input arrays, leading to long computation times and higher chances of over-fitting. The other is that in imaging, individual pixels are largely related to the pixels around them – this connection to surrounding pixels can be lost when collapsed into a 1D-array.

CNNs aim to fix these problems by looking at *regions* of the picture instead of *individual pixels*, and downsampling based on the data in these regions. This yields a new array that can be analysed quicker, and more effectively.

This report will introduce more in-depth theory in the **Theory and Algorithms** section, before outlining their implementations; any data preprocessing performed will be denoted in the **Method and Data** section. The results of said methods are subsequently presented and discussed in the following **Results** and **Discussion** sections. Finally, a **Conclusion** will summarize said work.

The study presented is a collaboration between S. H. Magnússon, G. S. Cabrera, and B. S. Dalen, and the machine learning scripts designed can be found at the following GitHub repository:

**https://www.github.com/GabrielSCabrera /MachineLearning/tree/master/FYS-STK4155 /Project_3**.

## II. THEORY AND ALGORITHMS

### A. Convolutional Neural Networks

Figure 1 illustrates the basic structure of a Convolutional Neural Network. The general idea is to process and downsample the input image in the convolution and pooling layers, and then analyse it using a regular neural network. We will now explain the structure of, and what happens in, each of the layers of a CNN.

#### 1. *Input Layers*

Firstly, we have the *input layer*: this is a grid of pixels representing the image to be studied. In the case of a colour picture, one might have multiple pixel grids, each representing a certain colour channel. The number of pixel grids is referred to as the image *depth*. If the input image is encoded with an RGB-value, for example, there would be a depth of *three*, representing red, green, and blue; in the case of a grayscale image, there would only be a depth of *one*. Since the EMNIST data set is grayscale the concept of depth will not be discussed further.

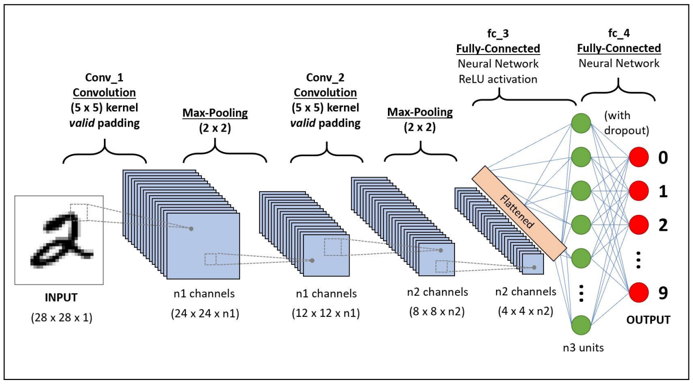

FIG. 1. Figure conceptualizing the Convolutional Neural Network structure for a grayscale image of a handwritten digit. Image taken from *Saha, S.* "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way" [7].

### 2. *Convolution Layers*

The next layer is the *convolution layer*: the main purpose of the convolution layer is to relate pixels to each other spacially. Here, we have an element that is referred to as a *filter* or *kernel* ($K$). This is a matrix, typically with shape $3 \times 3$ or $5 \times 5$.

Starting in the upper left corner, the algorithm looks at a section of the input layer ($I$) that has the same shape as $K$. It then performs a matrix multiplication between $I$ and $K$ and sums up all the elements in the resulting matrix. Next, the kernel is moved to the right by an amount referred to as the *stride length*, typically 1 or 2 – this process is then repeated until it reaches the rightmost section of the image. Here the algorithm shifts all the way back to the leftmost section, as well as shifting downwards by a number of pixels equal to the stride length.

This process is subsequently repeated until all sections of the image have been covered: the resulting output is then a new matrix with all the summations of the matrix multiplications, referred to as the *convolved feature*. This whole process can be repeated on the convolved feature if one wishes.

One might notice that the convolution layer will reduce the size of the input; for example, a $5 \times 5$ input with stride length 1 will result in a convolved feature with size $3 \times 3$ – this is not always desirable. One way to get around this is a process known as *padding*, which involves adding a layer of pixels of value 0 around the entire input. The $5 \times 5$ input will now instead be a $6 \times 6$ matrix, and the convolved feature will have size $5 \times 5$. This is know as *same padding*, given that the output size is now of equal size to the input; if the output has a larger size it is also known as same padding. In cases where stride length is larger than 1, padding can also be used to make the size of the convolved feature less reduced. This is know as *valid padding*. Typically both valid and same padding is used.

### 3. *Pooling Layers*

Next is the *pooling layer*: the pooling layer is used to reduce the size of the convolved feature. Once again, the algorithm selects one small portion of the input at a time, however it now returns either the *average* or *max value* of the section. Returning the average value is known as *average pooling*, while returning the max value is known as *max pooling*. Max pooling is better at reducing noise, and will therefore typically perform better.

After the convolution and pooling layers, the size of the convolved features should be reduced thoroughly. For larger images, it is typical to resend the feature though both layers several times, though this comes at a computational cost.

### 4. *Fully-Connected Layers and Output*

Finally we have the *fully-connected layer* (FCL), the simplest type of layer. Here the input is flattened, then sent through a standard feed-forward neural network layer. Since each node in this layer is connected to all the nodes in the next layer, it is called a fully-connected layer.

As with most NNs there are typically several FCLs, with the final layer referred to as the *output layer*. After analysing the output, backpropagation is applied to the fully connected layers, and over several epochs the NN will potentially be able to distinguish between different inputs.

## B. Extreme Gradient Boosting

XGBoost is a so-called *ensemble method*, where there are several predictors that contribute to a collective outcome. When applied to a classification problem, the XGBoost algorithm takes in a *base learner*, and subsequently builds on it to produce more accurate results. This is known as *forward stage-wise additive modeling*, and can produce exceptional results for both classification and regression problems.

The most common base learner used, and the one used throughout this study, is one consisting of poorly-performing *decision trees*; decision trees have a tuning parameters capable of producing models that attempt to predict an output for a given input. Figure 2 illustrates a decision tree conceptualization for effective time-management:

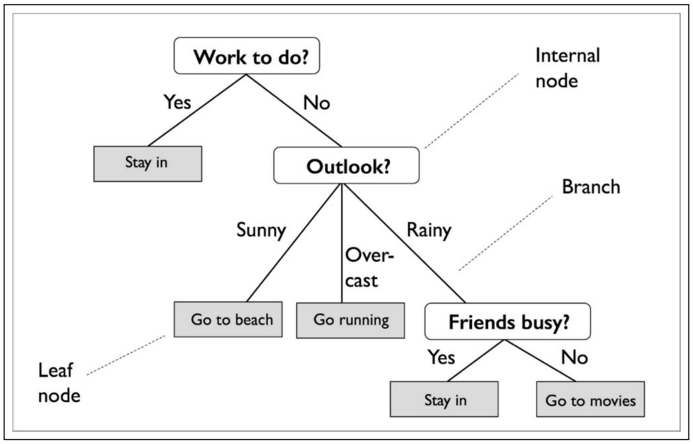

FIG. 2. Figure conceptualizing the decision tree method. This method is the one used as a base learner for the XGBoost algorithm. Image taken from *Li, Lorraine* “Classification and Regression Analysis with Decision Trees” [5].

This conceptualization is good at presenting the terminologies often used for decision trees.

The goal of any method (including the XGBoost method) is to create a predictor $f$ such that the loss function $L$ assigned to the method is minimized:

$$\hat{f} = \text{argmin}_{\hat{f}} \mathbb{E}_{x,y} \left[ L\left( y, \hat{f}(x) \right) \right] \tag{1}$$

The model must first be initialized by a base learner $\hat{f}_0$. To improve on this base learner, the following *Gradient Tree Boosting* algorithm is implemented for a certain number of iterations $M$ [4]:

1. Initialize $\hat{f}_0(x) = \text{argmin}_\gamma \sum_{i=1}^{n} L(y_i, \gamma)$
2. For $m = 1$ to $M$:
   (a) For $i = 1, 2, ..., N$ compute:
   $$r_{im} = \left[ \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f=\hat{f}_{m-1}}$$
   (b) Fit a regression tree to the targets $r_{im}$ giving terminal regions $R_{jm}$, $j =, 1, 2, ..., J_m$.
   (c) For $j = 1, 2, ..., J_m$ compute:
   $$\gamma_{jm} = \text{argmin}_\gamma \sum_{x_i \in R_{jm}} L\left( y_i, \hat{f}_{m-1}(x_i) + \gamma \right)$$
   (d) Update the model
   $$\hat{f}_m(x) = \hat{f}_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$$
3. Output $\hat{f}(x) = \hat{f}_M(x)$

There are several variables here to tune, though the main ones are the *learning rate* $\nu$ associated with the gradient descent and the *tree depths/sizes* $J_m$. These are the two hyper-parameters which will be optimized in the study. Besides these hyper-parameters, the loss function of the method must also be assigned.

## C. Evaluation Methods

To evaluate which model is the best of the two, some accuracy metrics are presented.

### 1. Accuracy Score

The accuracy score is the simplest accuracy metric presented, and is defined as

$$\text{Accuracy} = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\left( y_i = \hat{y}_i \right), \tag{2}$$

where $n$ is the number of testing points, $y_i$ is the actual value for a certain point $i$, and $\hat{y}_i$ is the predicted value. Additionally:

$$\mathbb{1}\left( y_i = \hat{y}_i \right) = \begin{cases} 1 & \text{for } \hat{y}_i = y_i \\ 0 & \text{else} \end{cases}. \tag{3}$$

An accuracy score of 1 means *all* predicted values are correct, while 0 means *none* of the predicted values are correct – a higher score is therefore desirable. The accuracy score is also sometimes represented as a percentage.

### 2. Confusion Matrix and Jaccard Index

The *confusion matrix* of a categorical predictor presents a more in-depth view of the method performance than that of the accuracy score. The confusion matrix better illustrates the misclassification specifics, namely which classes are more commonly misclassified. This is illustrated by the following three-class $A$, $B$, and $C$ confusion matrix:

| | | True | | |
|---|---|---|---|---|
| | | A | B | C |
| | A | AA | AB | AC |
| Pred | B | BA | BB | BC |
| | C | CA | CB | CC |

The non-diagonal elements represent the misclassifications of the predictions, e.g. element $CA$ counts how many predictions $C$ were made when the true value was $A$. In this definition, the prediction methods aim to diagonalize the confusion matrix, and maximize the parameters $AA$, $BB$, and $CC$ in relation to all the others. A measurement of the diagonality of the confusion matrix is presented in the so-called *Jaccard Index.*

The Jaccard index of a class $M$ is defined by:

$$J_{\text{M}} = \frac{\text{MM}}{\text{MM} + \sum_{\text{i} \neq \text{M}} (\text{Mi} + \text{iM})} \tag{4}$$

The numerator of the expression simply expresses the sum of all the non-diagonal elements which regard class $M$, where $iM$ is the confusion matrix value that measures how many times class $i$ is predicted when the true class was $M$. It is worth noting that if there are no diagonal elements relating to the M class $Mi = iM = 0$, then the class is perfectly predicted by the method and the Jaccard index becomes $J_{\mathrm{M}} = 1$. In the case of the confusion matrix presented above for three classes $A$, $B$, and $C$, then the average Jaccard index of the model performance is as follows [6]:

$$\begin{aligned} J &= \frac{1}{3}\left(J_{\mathrm{A}} + J_{\mathrm{B}} + J_{\mathrm{C}}\right) \\ &= \frac{1}{3}\Bigg(\frac{\mathrm{AA}}{\mathrm{AA}+\mathrm{AB}+\mathrm{AC}+\mathrm{BA}+\mathrm{CA}} \\ &\quad + \frac{\mathrm{BB}}{\mathrm{AB}+\mathrm{BA}+\mathrm{BB}+\mathrm{BC}+\mathrm{CB}} \\ &\quad + \frac{\mathrm{CC}}{\mathrm{AC}+\mathrm{BC}+\mathrm{CA}+\mathrm{CB}+\mathrm{CC}}\Bigg) \end{aligned} \tag{5}$$

The mean of the Jaccard index of the model is what will be used to assess models in relation to each other. The final method of assessment is to introduce a loss function which the functions can aim to minimize.

### 3. Zero-One Loss

The zero-one loss function is one commonly used for categorical cases, and is defined as

$$L_{\text{0-1}} = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}(y_i \neq \hat{y}_i) \tag{6}$$

This is the loss function which the methods presented aim to minimize, and can intuitively be seen as the inverse of the accuracy function.

## III. METHOD AND DATA

### A. Data Processing

Figure 3 illustrates some of the preprocessing which was performed on the dataset by the distributors of the data [2].

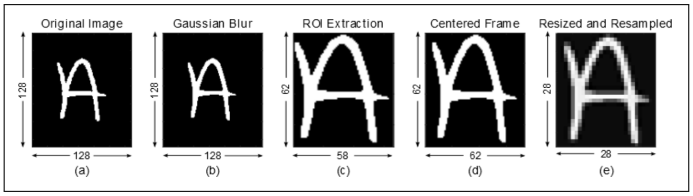

FIG. 3. Illustration of the steps taken to reduce the data size. Image taken from *Cohen, G.* "Emnist: an extension of mnist to handwritten letters" [2]

Figure 4 illustrates the balanced EMNIST data set distribution.

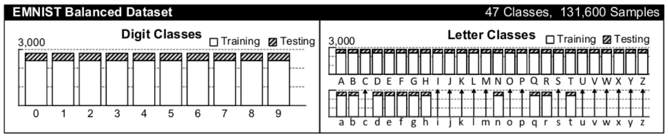

FIG. 4. Illustration of the distribution of the balanced EMNIST data set analysed. Image taken from *Cohen, G.* "Emnist: an extension of mnist to handwritten letters" [2]

The arrows pointing from one letter to the next indicate that the lowercase versions of the letters in question are considered indistinguishable from their uppercase counterparts. This means that the encoding used in the training and testing data is translated in the following manner:

$$\begin{array}{cccccccccc} [0, & 1, & 2, & 3, & 4, & 5, & 6, & 7, & 8, & 9, ... \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ [0, & 1, & 2, & 3, & 4, & 5, & 6, & 7, & 8, & 9, ... \end{array}$$

$$\begin{array}{cccccccccc} ...10, & 11, & 12, & 13, & 14, & 15, & 16, & 17, & 18, & 19, ... \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ ...A, & B, & C, & D, & E, & F, & G, & H, & I, & J, ... \end{array}$$

$$\begin{array}{cccccccccc} ...20, & 21, & 22, & 23, & 24, & 25, & 26, & 27, & 28, & 29, ... \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ ...K, & L, & M, & N, & O, & P, & Q, & R, & S, & T, ... \end{array}$$

$$\begin{array}{ccccccccc} ...30, & 31, & 32, & 33, & 34, & 35, & 36, & 37, & 38, ... \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ ...U, & V, & W, & X, & Y, & Z, & a, & b, & d, ... \end{array}$$

$$\begin{array}{cccccccc} ...39, & 40, & 41, & 42, & 43, & 44, & 45, & 46] \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ ...e, & f, & g, & h, & n, & q, & r, & t] \end{array}$$

#### 1. Outliers and Unfair Classes

A few interesting cases are found upon viewing and inspecting the data manually, some of which are outliers such as the one illustrated in Figure 5.

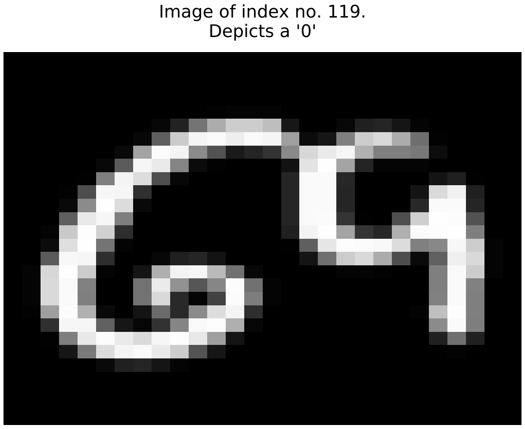

FIG. 5. Illustration of the 120th datapoint of the testing set. The pixels are labeled to be an image of a zero, though this is clearly not the case. The dataset creator seems to have inserted a so-called “easter egg” for the users.

This unusual case was found by looking over a relatively small number of datapoints by hand; there may therefore exist more such cases, potentially resulting in an unfair evaluation of the methods. Another factor which may cause an unfair method evaluation is the distinction between classes – the dataset is not written by an individual, but many. This causes the letters to vary depending on the writing styles of these individuals. This would be unproblematic had the methods included “style” information (e.g. in a one-hot encoded data column of “who” was writing), but this information is not disclosed in the data.

A final noteworthy factor of uncertainty is the general similarity between classes. These are classes that even a human would not be able to distinguish between, such as the similarities between the letter “O” and digit “0” categories, “q” and “9”, “I” and “1”, and so on. This produces a somewhat unfair evaluation of the model, as the issue is fundamentally related to the data. The goal here is therefore to generate a method which *only* produces misclassifications that seem reasonable to the human eye (i.e. a person could make the same mistakes). These misclassifications will be illustrated, and whether or not they are reasonable will be discussed later on.

## B. Method Implementation

### 1. Convolutional Neural Networks

The CNN portion of this report was implemented in `python` via the `keras` package, using *tensorflow* as a backend. Additionally, the `numpy` package was used extensively in each script. The scripts created each serve a separate purpose, and are run in sequence in order to perform the task at hand.

Firstly, the `.csv` files downloaded from the EMNIST source are read as `numpy` arrays, and are converted to smaller and more flexible `.npy` format for quicker reading. The data is then reloaded, and the features scaled to absolute variance and shifted such that their mean is set to zero. Additionally, *one-hot encoding* is implemented on the output data via the `keras` package.

Assuming there exist $N$ datapoints, each one a $28 \times 28$ image, the data is reshaped into a $N \times 28 \times 28 \times 1$ array, as required by `keras`. An arbitrary model is then created using the `keras.models.Sequential` object, and is trained. A test is then performed to make sure that saving/loading the model functions properly.

Finally, a range of hyper-parameters is selected, models are generated with these parameters, and the script is left to run overnight in order to determine how the configurations perform relative to each other.

Ultimately, four hyper-parameters are varied over:

- A $3 \times 3$ and $5 \times 5$ kernel.
- The activation functions *reLU* and *tanh*.
- Learning rates 0.001, 0.005, 0.01, 0.05, and 0.1.
- Batch sizes 32 and 64.

All combinations of the above are calculated, and the resulting accuracies are shown in Table I.

The selected configuration for the layers is the same for all models, and consists of two convolution layers of size 100, 66, with pooling after each layer. The second layer is subsequently flattened and a final fully-connected layer of size 46 is added, after which the output is obtained. It is also worth noting that the output activation function is always *softmax*, and that the number of epochs is always set to fifteen.

### 2. Extreme Gradient Boosting

The extreme gradient boosting algorithms were implemented via the “*XGBoost*” `python` library developed by the Distributed (Deep) Machine Learning Community (dmlc). The full public library can be found in the dmlc repository: https://github.com/dmlc/xgboost. This library has incredible functionality, and interfaces excellently with the `numpy`, `pandas`, and `scikit-learn` libraries - all of which are used in various portions of the script.

The main goal is to study which hyper-parameters produce the optimal XGB model performance; two parameter searches are therefore conducted in an attempt to optimize the tuning parameters of the XGBoost algorithm. The first of which is a variation of the learning rate, keeping all other parameters constant. The default learning rate parameter of the method is set to be $\nu = 0.5$, so the values around this are searched to see if there are any better variants. The first search simulated 20 different models using learning rates $\nu = \{0.05, 0.10, 0.15, ..., 0.45, 0.50, 0.55, ..., 0.95, 1.00\}$ to research the impact which the learning rate has to the model abilities.

The second grid search included the maximum tree depth $J_m$ as a variable in the search for which combination of values $\nu$ and $J_m$ produced the optimal model performance.

## IV. RESULTS

### A. Convolutional neural network

Table I summarizes the hyper-parameter grid-search used to determine the optimal configuration for the CNN model, ranked from best categorical accuracy to worst in descending order.

### B. Extreme Gradient Boosting

Figure 6 illustrates the learning rate analysis conducted keeping the rest of the parameters of the XGB constant (at their default values).

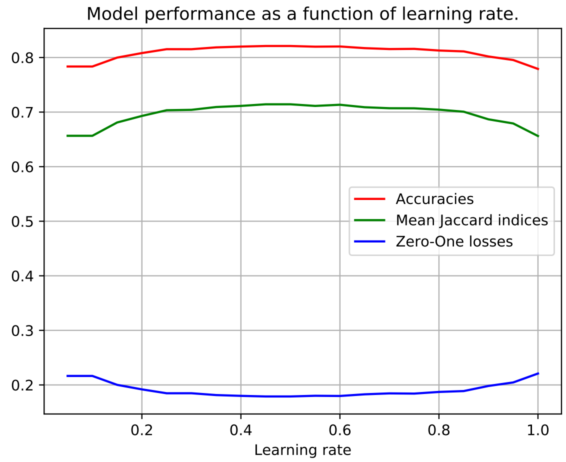

FIG. 6. A graph illustrating how the accuracy, mean Jaccard index, and zero-one loss performance evaluators vary depending on the learning rate. 20 models were calculated using learning rates $0.05, 0.1, 0.15, ..., 0.95, 1.0$.

Figures 7, 8 and 9 illustrate the accuracy, mean Jaccard index, and loss function results of the XGB grid-search, respectively.

| Accuracy | Activation | Learning Rate | Batch Size | Kernel Size |
|---|---|---|---|---|
| 0.8690 | tanh | 0.01 | 32 | 5 |
| 0.8690 | tanh | 0.01 | 64 | 5 |
| 0.8673 | tanh | 0.005 | 32 | 5 |
| 0.8670 | tanh | 0.01 | 32 | 3 |
| 0.8646 | tanh | 0.005 | 32 | 3 |
| 0.8634 | tanh | 0.01 | 64 | 3 |
| 0.8630 | tanh | 0.005 | 64 | 5 |
| 0.8613 | tanh | 0.05 | 64 | 3 |
| 0.8590 | relu | 0.001 | 32 | 3 |
| 0.8586 | relu | 0.001 | 32 | 5 |
| 0.8576 | relu | 0.005 | 32 | 3 |
| 0.8568 | tanh | 0.05 | 64 | 5 |
| 0.8537 | relu | 0.005 | 32 | 5 |
| 0.8531 | relu | 0.005 | 64 | 3 |
| 0.8529 | tanh | 0.005 | 64 | 3 |
| 0.8521 | tanh | 0.05 | 32 | 5 |
| 0.8504 | relu | 0.005 | 64 | 5 |
| 0.8504 | tanh | 0.05 | 32 | 3 |
| 0.8485 | relu | 0.001 | 64 | 5 |
| 0.8439 | tanh | 0.001 | 32 | 5 |
| 0.8419 | tanh | 0.1 | 64 | 5 |
| 0.8413 | relu | 0.001 | 64 | 3 |
| 0.8357 | tanh | 0.001 | 32 | 3 |
| 0.8271 | tanh | 0.1 | 64 | 3 |
| 0.8258 | tanh | 0.001 | 64 | 5 |
| 0.8245 | tanh | 0.1 | 32 | 5 |
| 0.8115 | tanh | 0.1 | 32 | 3 |
| 0.8103 | tanh | 0.001 | 64 | 3 |
| 0.7869 | relu | 0.01 | 64 | 3 |
| 0.7601 | relu | 0.01 | 64 | 5 |
| 0.7405 | relu | 0.01 | 32 | 5 |
| 0.0214 | relu | 0.05 | 64 | 5 |
| 0.0213 | relu | 0.1 | 32 | 5 |
| 0.0213 | relu | 0.05 | 32 | 5 |
| 0.0213 | relu | 0.1 | 32 | 3 |
| 0.0213 | relu | 0.05 | 64 | 3 |
| 0.0213 | relu | 0.05 | 32 | 3 |
| 0.0213 | relu | 0.01 | 32 | 3 |
| 0.0213 | relu | 0.1 | 64 | 5 |
| 0.0213 | relu | 0.1 | 64 | 3 |

TABLE I. Ranked categorical accuracies for the CNN with varying hyper-parameters; for a constant layer configuration [100, 66], 15 epochs, and output activation function softmax.

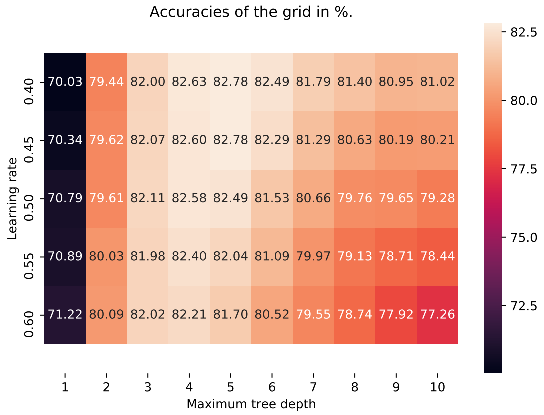

FIG. 7. A heatmap illustrating a grid search over the accuracies of the XGB models as a function of the learning rate and max tree depths.

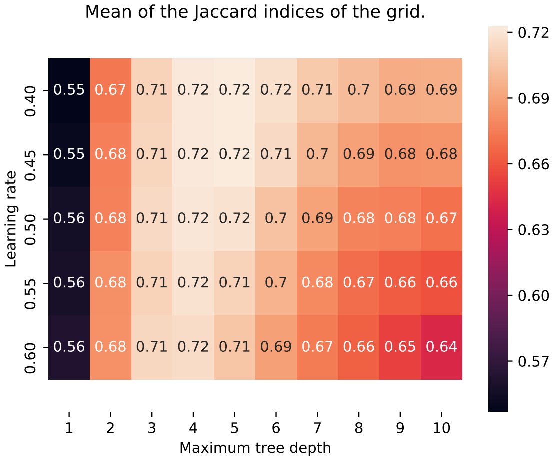

FIG. 8. A heatmap illustrating a grid search over the mean Jaccard indices of the XGB models as a function of the learning rate and max tree depths.

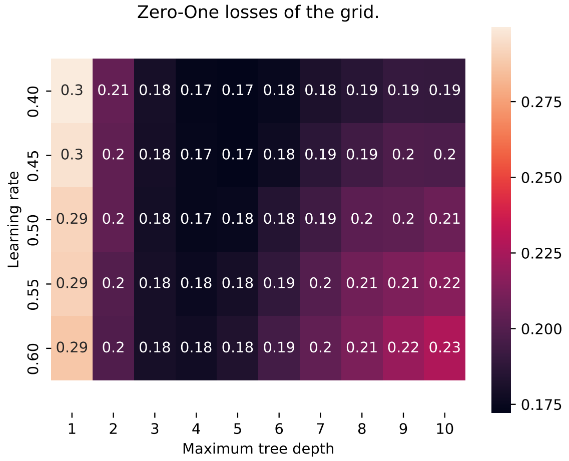

FIG. 9. A heatmap illustrating a grid search over the *zero-one* loss functions of the XGB models as a function of the learning rate and max tree depth.

Figure 10 illustrates some of the misclassifications of the optimal XGB algorithm.

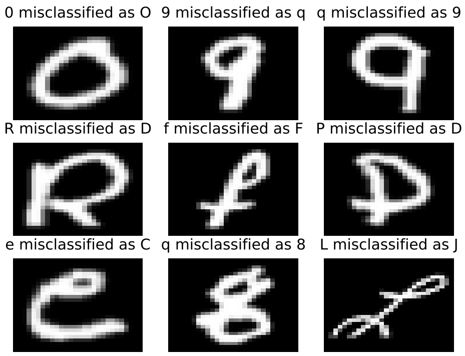

FIG. 10. Illustration of a random set of misclassifications made by the optimal XGB model.

### C. Summary

Table II compares the best results found in each model to each other, and those found in *Cohen, G. et al.* “Emnist: an extension of mnist to handwritten letters.”

| Method | Accuracy | Remarks |
|---|---|---|
| OPIUM | $78.02\% \pm 0.92\%$ | Results of Cohen, G. [2] |
| CNN | 86.90% | Best config. in Table I |
| XGB | 82.78% | Best config. in Figure 7 |

TABLE II. Table listing the accuracies of the two methods presented (CNN and XGB) as well as the OPIUM method benchmark by *Cohen, G.* [2].

## V. DISCUSSION

### A. Convolutional Neural Network

As seen in Table I, varying over four hyper-parameters raises some interesting points. Firstly, it is clear that using *tanh* as the activation function in the hidden layers yields better results than $reLU$; note that the eight best accuracy scores all use *tanh* as their activations, while the twelve worst scores all have $reLU$ activations.

Secondly, it was determined that a learning rate in the range $0.005 \rightarrow 0.05$ led to the best accuracy scores, while anything larger greatly under-performed: overall, a value around 0.01 appeared to be ideal.

Next, the kernel size did not appear to greatly affect the functionality of the CNN, though a $5 \times 5$ kernel *did* still perform slightly better than the $3 \times 3$ kernel.

Finally, there was little difference found between a batch size of 32 and 64; this is seen at the top of the table, where the two best models both have the same accuracies and parameters, with exception to their differing batch sizes.

On a final note, while the two "*best*" models both perform equally well, it would be preferable to use the second one listed – the reason being that a larger batch size allows for greater parallelization, and could therefore be more computationally efficient when using a GPU or computing cluster.

### B. Extreme Gradient Boosting

Figure 6 illustrates the impact which the learning rate has on the prediction accuracy of the balanced EMNIST data set.

Figures 7, 8, and 9 illustrate the performances of the 50 models created in the grid search. The XGB algorithm has an accuracy maximum for tree depth $M = 5$ and a learning rate of either $\nu = 0.40 \vee 0.45$. The same maximum seems to apply to the mean of the Jaccard index evaluation, though the decimal places are not sufficient to see. The slight color difference between the 0.72 values reveals that there was a similar maximum found at hyper-parameters $M = 5$ and $\nu = 0.40 \vee 0.45$. These optimal parameters produced a minimum of the loss function to be 0.17, where the rounding once again does not allow this to be displayed with better precision in Figure 9.

Additionally, Figure 10 illustrates a random set of the misclassifications of the best XGB classifier. This figure illustrates the difficulties inherent to the dataset, given that some of the classifications could not be performed by a human. Classifications of "O"s versus zeros are always going to be tricky. The same goes for any "I"s and ones, or "q"s and nines, among others.

## VI. CONCLUSION

Both the convolutional neural network and extreme gradient boosting methods were found to outperform the OPIUM method presented by Cohen, G.

The CNN scheme was found to produce the most accurate predictions of the balanced EMNIST dataset at a correct classification rate of 86.90% using the configuration listed at the top of Table I. The XGB algorithm was found to produce a correct classification rate of 82.78% using a tree depth of 5 and a learning rate of 0.40 or 0.45.

The inaccuracies of the methods were found to be largely due to characteristics of the data, rather than poor performance of prediction methods. The balanced EMNIST data set is tricky in that several classes which are indistinguishable (e.g. zeros and O's, I's and ones etc.), as well as a few outliers (such as the one illustrated in Figure 5) were included. The data was also found to include various fonts and hand-writing styles, something which the methods were not aware of. These factors all provided a dataset which provided a large difficulty of the digit recognition methods.

All things considered, it would be interesting to attempt implementing further improvements on the 86.90% accuracy result of the CNN, as well as seeing whether or not the XGB algorithm can come closer by further tuning of parameters.

## REFERENCES

[1] Bojarski, M., Testa, D. D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J. and Zieba, K. [2016], 'End to end learning for self-driving cars'.

[2] Cohen, G., Afshar, S., Tapson, J. and van Schaik, A. [2017], 'Emnist: an extension of mnist to handwritten letters'.

[3] Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M. and Thrun, S. [2017], 'Dermatologist-level classification of skin cancer with deep neural networks', *Nature* **542**(7639), 115–118.
**URL:** *https://doi.org/10.1038/nature21056*

[4] Hastie, T., Tibshirani, R. and Friedman, J. [2001], *The Elements of Statistical Learning*, Springer Series in Statistics, Springer New York Inc., New York, NY, USA.

[5] Li, L. [2019], 'Classification and regression analysis with decision trees'. [Online; posted 15-May-2019, accessed 15-December-2019].
**URL:** *https://towardsdatascience.com/https-medium-com-lorrli-classification-and-regression-analysis-with-decision-trees-c43cdbc58054*

[6] Li, L., Wu, Y. and Ye, M. [2015], 'Experimental comparisons of multi-class classifiers', *Informatica* **39**(1).

[7] Saha, S. [2018], 'A comprehensive guide to convolutional neural networks — the eli5 way'. [Online; posted 15-December-2018, accessed 11-December-2019].
**URL:** *https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53*