English-Arabic Handwritten Character Recognition using Convolutional Neural Networks

Abdijabar Mohamed
Middlebury College
Middlebury, VT
aymohamed@middlebury.edu

Ella Rohm-Ensing
Middlebury College
Middlebury, VT
erohmensing@middlebury.edu

ABSTRACT

Handwritten character recognition (HCR), a branch of Optical Character Recognition (OCR), is an emerging and active research area with far-reaching applications in many fields. The recognition complexity varies across the diverse human languages due to variations in the arrangement, shapes and the total number of characters. There are numerous works that are dedicated to English Handwritten Character Recognition (EHCR). However, relatively little attention has been paid to the Arabic language despite it being one of the world's most spoken languages. In recent years, deep learning techniques have attained a better state-of-the-art performance in the area of OCR than traditional machine learning methods. The class of convolutional Neural Networks (CNN or ConvNet), a biologically-inspired class of deep learning, has achieved great success in the recognition of handwritten English and Arabic characters. In this project, we present a deep neural network for the handwritten English/Arabic character recognition problem that is composed of four CNN models, some of which employed regularization parameters such as Dropout and Early Stopping to prevent overfitting. Our deep learning model for recognizing English letters was trained on 124,000 samples and validated on 20,800 samples from the EMNIST Letters dataset, achieving a test accuracy of 93.63%. In the case of the model for recognizing English handwritten digits, we trained and tested on 60,000 and 10,000 samples respectively from the MNIST dataset, obtaining a test accuracy of 99.15%. For the Arabic letters recognition model, we trained on 13,439 samples and tested on 3,359 samples from the Kaggle Handwritten Arabic Letters dataset, attaining a test accuracy of 84.99%. Last but not least, the model that recognizes handwritten Arabic digits was trained on 60,000 samples and tested on 10,000 samples from the MADBase Arabic handwritten digits database, with a test accuracy of 99.68%.

Keywords

Convolutional Neural Network (CNN); Deep Learning (DL); Optical Character Recognition (OCR); Handwritten Character Recognition (HCR); EMNIST; MNIST; MADBase; Kaggle

1. INTRODUCTION

OCR is an research area that has practical application in numerous fields such scanners used at store checkout counters, bank cheques, and automation in the postal system [17]. HCR is a branch of OCR and is the subject at hand in this work. The handwritten character recognition process is faced with the fundamental challenges of the enormous varieties of handwritten scripts that can be attributed to difference in writing styles among different people and the recognition of unconstrained cursive handwritings [16]. Moreover, in the case of the Arabic characters, the recognition problem is more complicated. This is attributable to the fact that many characters have similar shapes with differences emanating from the variations of the position of dots relative to the main part of the character [18].

It is important to mention that the current attention that researchers accord the recognition of handwritten Arabic characters vis-a-vis the place of Arabic as one of the most widely spoken languages do not commensurate. The Arabic language belongs to the "Semitic" group of languages hence it is closely related to such languages as Hebrew, Aramaic, Amharic and Tigrinya. It is spoken by approximately more than 420 million people in the Middle East, North Africa, the Horn of Africa and in other parts of the Arab world thus making it the 6th most spoken language in the world. The Arabic alphabet (also known as arabic abjad) Similar to the Hebrew is cursively written from right to left and consists of 28 letters and 10 digits.

Deep Learning is a class of Machine Learning that uses a hierarchical level of Artificial Neural Networks (ANN) inspired by the structure of the neurons in the human brain. The word "deep" as invoked here defines the multilayer architecture of these networks that usually consist of multiple "hidden layers" of nodes between the input and output nodes. Its methods learn feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features. They include learning techniques for a wide array of deep architectures such as neural networks with numerous hidden layers [8].

In the Big Data age, rapid and revolutionary enhancements of image recognition is being witnessed due to state-of-the art results provided by Convnets, a type of Deep Neural Network (DNN). The success of CNNs is attributable to their capability to learn from high-dimensional complex inputs and non-linear mappings from big data. Besides, CNNs support parameter sharing -sharing of weights by all neurons in a particular feature map- and local connectivity (each neuron is connected only to a subset of the input image), techniques that reduce the number of parameters in the system thus boosting the overall computation of the system [13].

There are a number of DL frameworks that provide building blocks for designing, training, testing and validating deep

neural networks through a high level programming interface [3]. In this project, we used Keras, a high-level neural networks API built on TensorFlow, a machine intelligence library that utilizes powerful computational graphs. Keras was chosen to be our go-to framework since it uses Python programming language, which arguably makes writing scripts easier than in native TensorFlow codes. As a result, in this paper we propose a robust multimodal convolutional deep learning system for solving the problem of EHCR and AHCR using Keras with TensorFlow. By virtue of employing ConvNets, this system aims to outperform the conventional EHCR and AHCR.

The contribution of this project can be encapsulated as follows: propose robust ConvNet DL architectures for solving the subclasses of the EHCR and AHCR problems, attain the highest accuracy on the AHCR problem and contribute to the less researched area of AHCR.

The rest of this paper is structurally organized as follows: Section 2 describes the problem statement of our project, whereas Section 3 gives a review of the some of the related work done in the area under consideration. Sections 4 and 5 detail our experiment's techniques and the obtained results. Section 6 entails a discussion of our work's methods and findings while Section 7 is an acknowledgement of all those whose help was crucial to the success of our project.

2. PROBLEM STATEMENT

The motivation of this project is to develop a robust multimodal Deep CNN system that is able to improve on the test accuracies (or chart the paths for possible exploration in that direction) of existing systems by utilizing a variety of datasets such as Kaggle available MADBase datasets, EMNIST, MNIST and our own data. Moreover, this work aims at contributing to solving the recognition challenge that faces the Arabic alpha-numerals due to little attention from the Deep Learning scholarly community.

In this work, we apply a number of techniques that we noticed that previous researchers did not utilize. For instance, in some of our architectures, we use regularization techniques such as Early Stopping, technique for controlling overfitting in CNN by stopping the training before the weights have converged. Besides, some of the architectures that those who previously worked on this problem used simple layers such as in Das et al's MLP architecture. By using Keras sequential model, our work uses a range of systems with varying levels of layers complexity providing us with the freedom to explore varieties of architectures.

In order to avoid the difficulties posed by the problem of latency in data processing, this project utilizes Gattaca, a 16-core (dual socket) Linux workstation with 128 GB of RAM and a NVIDIA Titan Xp GPU [4]. As a result of the user privileges we had in Gattaca that enabled us to install GPU-enabled TensorfFlow with the necessary support of the already available CUDA tools for acceleration, we assumed that it was prudent to choose Keras (with TensorfFlow) as the developing environment.

3. RELATED WORK

Recently, research is being directed towards the improvement of character recognition. In the English alphabets recognition task, Perwej et al devised a simple CNN scheme for recognizing and predicting on human handwritten alphabets attaining an average test accuracy of 82.5% [12]. We opine that although this paper cited more than 30 times (as observable from Google Scholar citations), the apparent poor performance is due to the lack of proper activation functions, regularization techniques. Our deep CNN system overcomes this challenge since it has the aforementioned techniques and attains a test accuracy of 93.63%. It is important to bear in mind that while Perwej et al used lowercase letters, our approach uses EMNIST Letters (consists of 145,600 characters and 26 balanced classes). In the English numerals recognition problem, Saeed implemented a Multilayer Perceptron (MLP) Neural Network to recognize and predict handwritten digits using MNIST dataset. Their proposed MLP system attained an overall accuracy of 99.32%. They achieved this impressive accuracy despite using a mere 5000 data samples from the entire MNIST database [15]. In this project, we similarly use the MNIST data but employ a deep CNN for training and testing on the data that achieved a testing accuracy of 99.15%. Our model is trained on 60,000 samples and tested on 10,00. We believe that a possible, robust CNN-MLP classifier with a larger and improved data can achieve perhaps better results than both of our and Saeed's system.

In the area of recognition of Arabic alphabets, El-Sawi et al. built a CNN DL architecture that upon being trained and tested on the Arabic Handwritten Character Recognition (AHCR) achieved an accuracy of 94.9% on the testing data [6]. Similarly, Das et al. has pioneered in the research of the recognition of handwritten Arabic digits. They developed a Multilayer Perceptron (MLP) based pattern classifier for the handwritten Arabic numerals and subsequently obtained an accuracy of 94.93%. The challenge that Das et al faced is the then absence of standardized data for feeding into their models [5]. Our deep CNN for deciphering handwritten Arabic numerals overcomes this by utilizing the large binary Arabic handwritten MADBase Arabic digits database that is available on Kaggle website.

4. METHODS

4.1 Self-Populating an English Dataset

The English character dataset that we self-populated was created from online fonts, due to lack of access to large amounts of raw handwritten data. In addition, we hoped that as a jumping off point, fonts, which tend to be more regulated than a person's handwriting, would have more detectable patterns as a whole to be picked up on by a convolutional neural network. In total, we processed over 3,500 fonts to create originally over 150,000 samples for training and testing. This set was pared down to just over 115,000 samples after data cleaning for better accuracy. The process of turning fonts into preprocessed data samples ready for CNN training or testing consisted of three steps: download, sample generation, and sample preprocessing.

4.1.1 Font Download

Our font samples were downloaded from www.dafont.com, an online repository that currently supports the download of 34,905 fonts. These fonts vary widely, and many do not resemble the normal handwriting of any person, so we narrowed our download to the "basic" and "script" subsections.

The automated download of TrueType (.ttf) files was achieved using Downloader.R. In this R file, we used the "rvest" pack-

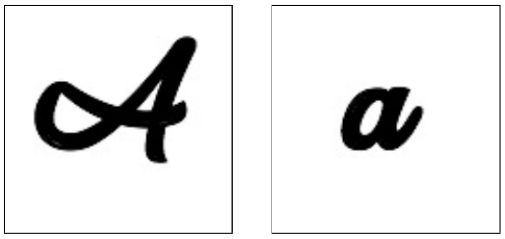

Figure 1: Sample outputs of parser.py.

Figure 2: Sample outputs of eng_preprocessor.py.

age, for its ease of manipulating HTML code [2]. By using this package, we were able to search each page's HTML code for download nodes, which contained download links in their href attributes. After creating a list of these download links, we used R's native download.file() method to download the .zip files associated with each font, as well as other native file and directory manipulation methods to open the .zip files, copy the TrueType files to the correct font folder, and delete extra files. This allows the automated download to stay a clean method throughout without building up extra files which would require more storage space and manual deletion at the end of the process. In addition, we also used this file to manipulate the names of our font TrueType files, deleting special characters such as spaces and dashes in order to ensure that the later LaTeX files that call these fonts did not have any trouble processing them.

4.1.2 Generating Sample Images in LaTeX

For each font, we needed to generate a different image for each lowercase alphabetic character, capital alphabetic character, and number. We decided to generate these images using LaTeX because of the ability to write and run the LaTeX code through Python and LaTeX's ease of manipulation of output formatting. Our Python file parser.py achieved the automation of this step of the process.

For each TrueType font in the font folder we populated in the font download step, parser.py loops over each of the 62 characters, creating a LaTeX file for each. In this LaTeX code, the document class is set to test.cls, a LaTeX class file hidden in the build folder that sets font size and document size and calls the necessary libraries for setting the font. Then the code sets the font to the current font in the loop, prints the current character in the center, and outputs the resulting .pdf file to the build folder.

After each .pdf is outputted to the build folder, it is moved to our output folder, where it is converted to .jpg using command line arguments from the "os" package in Python. It was important that files be converted as they are created from .pdf to .jpg, instead of batch conversion upon completion, because the amount of data would have grown immensely large with a build-up of .pdf files.

4.1.3 Data Pre-Processing

Output .jpgs generated from parser.py were black-on-white, and although font size was regulated, the actual size of the characters were varied. In order to remedy this, we wrote a Python preprocessing file that takes in these varied images and regulates them in order to make it easier for the convolutional neural network to recognize patterns in them (see figure x).

The file eng_preprocessor.py first loops over the folder of generated .jpg files from step 4.1.2., counting the files that are not blank, and deleting lowercase characters that resemble their capital counterparts (reasoning for this is given in the discussion section). After counting the total number of samples, it creates empty arrays for training data (containing 6/7 of the data) and testing data (1/7). These arrays are of width 785, where the 0th index encodes the value of the character (see Table 1) and the 1st through 784th indices contain pixel information for the 28x28 size images.

Preprocessing of each image started with filtering out any blank images, i.e. the image was only added to the training or testing set if it contained some black pixels. The necessity of this step arose as it became clear that some fonts were generating blank samples for characters they did not support. More information about issues with respect to data cleaning appear under Discussion.

After checking the existence of a character, Python's image manipulation "cv2" package was used to resize the .jpg to 28x28 pixels, and inverts the images to white on a black background. This choice and many further choices with respect to preprocessing were made to attempt to emulate the MNIST dataset, a reputable dataset on which many machine learning methods are able to predict classification with high accuracy.

From here, the preprocessor centers the character by removing any empty bordering rows and columns, and resizing the character to be a 20x20 square on a 28x28 background. Lastly, shift methods are used to move the character so that the center of mass of the character, not just the center of the pixels in the character, is the true center.

This centered, size-regulated image has its contrast heightened as far as possible, using a threshold function in cv2 to send any grays to whichever of white or black is closest.

Class	Char	Class	Char	Class	Char	Class	Char
0	0	12	C	24	O	36	a
1	1	13	D	25	P	37	b
2	2	14	E	26	Q	38	d
3	3	15	F	27	R	39	e
4	4	16	G	28	S	40	f
5	5	17	H	29	T	41	g
6	6	18	I	30	U	42	h
7	7	19	J	31	V	43	n
8	8	20	K	32	W	44	q
9	9	21	L	33	X	45	r
10	A	22	M	34	Y	46	t
11	B	23	N	35	Z		

Table 1: Data class labels for self-generated data.

The finished images are saved into the premade testing and training arrays, and those arrays are saved into .csv files for input into a convolutional neural network.

4.2 Other Databases

4.2.1 MNIST Digits

This benchmark dataset that is accessible through Keras functionality is a subset of a larger set available from National Institute of Standards and Technology (NIST) and is made up of a training set of 60,000 examples and a test set of 10,000 examples [9].

4.2.2 EMNIST Letters

A dataset comprising of 145,600 characters that merges a balanced set of the lowercase and uppercase letters into a single 26-class task [7]. This dataset exists as 28x28 pixel image formats and is formatted in the same manner as the MNIST dataset.

4.2.3 MADBase Database

This database contains Arabic handwritten digit images and consists of 60000 training and 10000 testing images [11].

4.2.4 Kaggle Arabic Letters

This database, found under the "Arabic Handwritten Characters Dataset" on Kaggle, consists of 13,440 training data and 3,360 test data of size 32x32 [10].

4.3 CNN Architchture

CNNs are feedforward networks in that information flow occurs only unidirectionally, from their inputs to their outputs. They are inspired by the attempt to mimic the human intelligence and it draws parallels to the visual cortex in the brain, which consists of alternating layers of simple and complex cells [14]. A CNN is capable of converting the input image structure through each particular layer of the network in order to extract automatically the features of the input image. The typical CNN with streaming in data is composed of the input data Convolutional Layers (the layers that manipulate the image upon feeding it), Fully Connected Layers and the Output [14]. Figure 3 below shows the general topology of a CNN for recognizing character(s).

4.3.1 Input Data

The input to a convolutional layer is usually in the form of M by M by C where M is the height and the width of the image, M × M is the number of pixels in the image and C refers to the number of channels per pixel. For RGB images, C is equivalent to 3 while in Gray scale images, C is equal to 1.

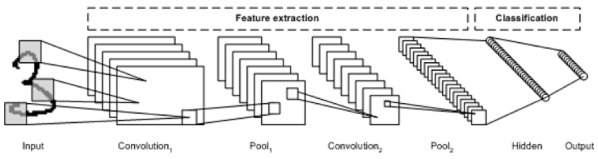

Figure 3: Chart of accuracy incease over epochs

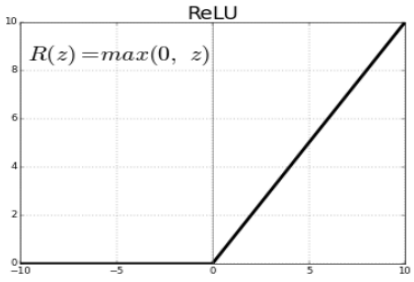

Figure 4: The ReLU function.

4.3.2 Convolutional Layers

A convolutional layer is simply the input image feature extractor. During feature extraction, the CNN uses a matrix called filter to locate the features of the image. Each filter corresponds to a particular feature. The filter is slid across the image pixels and after computation, a convolved output is produced. The convolution procedure is as illustrated in Figure 5 below.

The activation functions used in our project are Rectified Linear Unit (ReLU) and Softmax. ReLu is a form of activation which is simply an activation that is thresholded at zero (as shown in Figure 4) while Softmax (used in the output of our system) is an activation technique used for solving multi-class problems as in our case. We chose ReLu since it reduces the likelihood of the gradient to vanish hence preferably selected over such activation methods as the Sigmoid function. Likewise, we chose the Softmax activation function since it probabilistically determines the outputted prediction making it easier to work with in the process of attempting to increase the accuracy.

4.3.3 Pooling Layers

Usually, pooling layers occur after a convolutional layer. Pooling basically entails manipulating rectangular blocks of the image pixels and subsampling to obtain a single output from that given block. Examples of pooling include maximum pooling, average pooling etc. In this project, we use maximum pooling since it extracts the most salient features like edges hence our preference for it over average pooling.

4.3.4 Fully-Connected Layers

A fully connected layer takes all neurons in the previous layer (convolutional, fully connected etc) and connects it to every single neuron in a flattened vector-like structure.

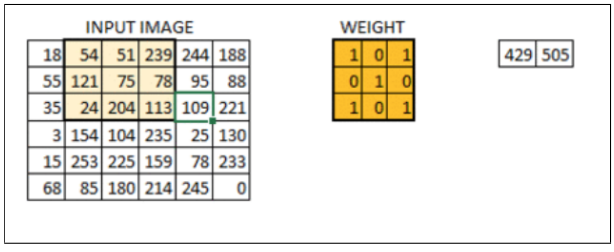

Figure 5: The convolution process.

4.3.5 Output

Finally, in the output of the CNN, we obtain predictions of our image data into a class of the output classes that were determined in the labeling of the data.

4.3.6 Model Structures

CNN Structure - Arabic Digit Recognition			
Layer	Layer Type	Output Shape	# of Parameters
1	Conv 2D	(None, 28, 28, 64)	640
2	Batch Norm	(None, 28, 28, 64)	256
3	Activation	(None, 28, 28, 64)	0
4	Conv 2D	(None, 27, 27, 32)	8224
5	Batch Norm	(None, 27, 27, 32)	128
6	Activation	(None, 27, 27, 32)	0
7	Flatten	(None, 23328)	0
8	Dense	(None, 10)	233290
Total Parameters: 242,538		Trainable Parameters: 242,346	

CNN Structure - Arabic Letter Recognition			
Layer	Layer Type	Output Shape	# of Parameters
1	Conv 2D	(None, 30, 30, 32)	320
2	Max Pooling 2D	(None, 14, 14, 0)	18,496
3	Conv 2D	(None, 12, 12, 64)	18496
4	Conv 2D	(None, 10, 10, 128)	73856
5	Max Pooling 2D	(None, 5, 5, 128)	0
6	Dropout	(None, 5, 5, 128)	0
7	Conv 2D	(None, 4, 4, 256)	131328
8	Max Pooling 2D	(None, 4, 4, 256)	0
9	Flatten	(None, 4096)	0
10	Dense	(None, 128)	524416
11	Dropout	(None, 128)	0
12	Dense	(None, 29)	3741
Total Parameters: 752,157		Trainable Parameters: 752,157	

CNN Structure - English Digit Recognition			
Layer	Layer Type	Output Shape	# of Parameters
1	Conv 2D	(None, 26, 26, 32)	320
2	Conv 2D	(None, 24, 24, 64)	18,496
3	Max Pooling 2D	(None, 12, 12, 64)	0
4	Dropout	(None, 12, 12, 64)	0
5	Flatten	(None, 9216)	0
6	Dense	(None, 128)	1,179,776
7	Dropout	(None, 128)	0
8	Dense	(None, 10)	1290
Total Parameters: 1,199,882		Trainable Parameters: 1,199,882	

CNN Structure - English Letter Recognition			
Layer	Layer Type	Output Shape	# of Parameters
1	Conv 2D	(None, 26, 26, 32)	320
2	Conv 2D	(None, 24, 24, 64)	18,496
3	Max Pooling 2D	(None, 12, 12, 64)	0
4	Dropout	(None, 12, 12, 64)	0
5	Flatten	(None, 9216)	0
6	Dense	(None, 128)	1,179,776
7	Dropout	(None, 128)	0
8	Dense	(None, 27)	3483
Total Parameters: 1,202,075		Trainable Parameters: 1,202,075	

Table 2: Layer structures of our CNN models.

5. RESULTS

In generating our self-populated dataset, we downloaded over 3,000 fonts. Afte preprocessing these fonts and filtering out visible and easily filterable bad data, we finished with 111,299 pre-processed samples. With respect to training a model on this data, we succeeded in preprocessing the data in such a way that it could be fed to a convolutional neural network for testing and training. In total, we trained the model on 95,000 samples and tested it on 16,299 samples. The highest accuracy we obtained on this dataset was 71.8%, which was not as high as we had hoped. Suggested reasons for this low accuracy are discussed in the Discussion section.

In order to enhance the performance of the models trained on the four datasets from outside sources, we tweaked the hyperparameters of the models for each of the respective subproblems. Due to the limited scope of this paper and bearing in mind that our model architectures are quite similar with minor modifications, in this section we analyze the results of the training experiment that we conducted for the model that aims to recognize Arabic handwritten letters.

The final hyperparameters that the model for recognizing Arabic letters used are initial learning rate that is equivalent to 0.1, 9 epochs and a batch size of 50. Epoch refers to a complete pass over through a particular dataset whereas a batch is the total number of training examples in one pass. As shown in figure 6, both the training and validation accuracies increase with increase in the number of epochs. After approximately 7 epochs, the increase in these accuracies reach a 'plateau in performance' stage beyond which an increase in the number of epochs does not improve the performance. Similarly, the training loss and validation with increase in the number of epochs until 7 epochs are attained after which both of the losses level. Loss values measure how well or poor the model performs after each given optimization.

In aiding the model to find the optimal learning rate, we employed step decay, one of the most popular forms of learning rate annealing where the rate is decreased by certain percentage after a particular number of training epochs (hence the use of the term decay). In our case, we defined a learning rate schedule in which the learning rate (where initial rate = 0.1) is updated during the training as specified by the drop functionalities encapsulated in this schedule. The aforesaid learning schedule is then integrated into our model as a callback, a set of functions to be applied at a given stage during the training process [1].

The constituent models of our system achieved promising results. The model for recognizing English digits attained a test accuracy of 99.15% , while the one for deciphering English handwritten letters obtained a test accuracy of 93.69%.

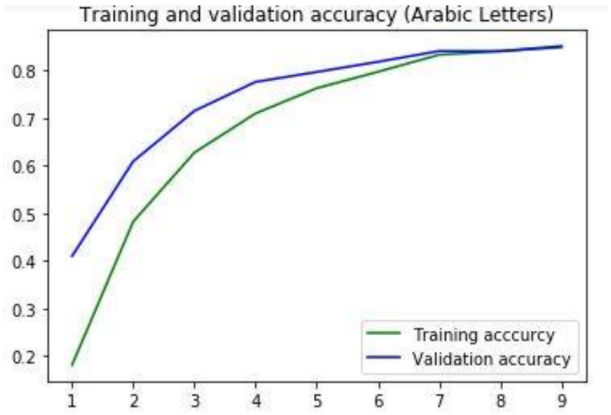

Figure 6: Chart of accuracy incease over epochs

Similarly, the model for the Arabic digit recognition achieved a test accuracy of 99.68% while the one that recognizes handwritten Arabic letters attained a test accuracy of 84.99%.

6. DISCUSSION

Handwritten Character Recognition for the world's languages is an emerging area thus necessitating constant improvement of the accuracies of such systems. In this project work, we propose a deep, convolutional neural network system for recognizing handwritten characters in both the English and Arabic languages. Besides contributing to the improvement of the accuracy of such systems that are geared towards Arabic, our work paves the way for possible improvements to the accuracy of the English character recognition problem. The latter can be attained by the combining our proposed architectures with those the past researchers as detailed in section 3.

Despite the success of our system, we believe that its accuracy could have been improved if we integrated two important tools of optimization that are currently gaining popularity among Deep Learning enthusiasts. These tools are TensorFlow.js, a browse-based JavaScript library for training models and Tensorboard, a visualization tool for models. By using TensorFlow.js, one is able to define and train DL models and by utilizing Tensorboard, one can readily understand and debug the model through the browser.

In addition to optimizing models for recognition of datasets from outside sources, we explored what it takes to create a data pipeline for preprocessing data that is optimized for machine learning. In this process, we found that the amount of unclean data is higher than one would expect, and is particularly hard to filter. Some issues we encountered and overcame within the preprocessing step were blank samples and the similarity between some capital and lowercase letters after preprocessing. For example, after image resizing, a capital 'R' and a lowercase 'r' are distinguishable, but a capital 'C' and a lowercase 'c' are often not. To remedy this problem, we decided to limit our 62 classes to 46, as described in Table 1. This labelling was inspired by the labeling of the EMNIST dataset [7], and greatly increased our accuracy levels. Other issues that we came across in data cleaning included:

- Some character forms are completely different in cursive fonts vs. printed fonts, for example, the letter 'r'
- Some fonts dd not support some characters, but generated pictures instead of blank images
- Some fonts mapped both capital and lowercase letters to capital letters, or both to lowercase letters

Considering future work, we would like to continue improving the accuracies of our system and applying the proposed frameworks to more complicated languages such as Chinese, Japanese and Amharic.

7. ACKNOWLEDGMENTS

We would like to sincerely thank Professor Jason Grant for his suggestions, feedback, advice, and general mentorship throughout the course of this project. Additionally, we would like to express our gratitude to Professor Michael Linderman for providing us with remote user access to his Gattaca Linux station for storing project files and availing the massive parallel computation power necessary for running our deep learning system scripts.

8. REFERENCES

[1] Create a callback. https://keras.io/callbacks/.

[2] Package 'rvest'. https://cran.r-project.org/web/packages/rvest/rvest.pdf, 2016.

[3] Deep learning frameworks. https://developer.nvidia.com/deep-learning-frameworks, 2018.

[4] Working on gattaca. https://slack-files.com/T3QSJKAQ6-F60VA9ERM-110260a69e, 2018.

[5] N. Das, A. F. Mollah, S. Saha, and S. S. Haque. Handwritten arabic numeral recognition using a multi layer perceptron. *arXiv preprint arXiv:1003.1891*, 2010.

[6] A. El-Sawy, M. Loey, and E. Hazem. Arabic handwritten characters recognition using convolutional neural network. *WSEAS Transactions on Computer Research*, 5:11–19, 2017.

[7] P. Flanagan. The emnist dataset. https://www.nist.gov/itl/iad/image-group/emnist-dataset, 2017.

[8] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[9] Y. LeCun. The mnist database of handwritten digits. http://yann. lecun. com/exdb/mnist/, 1998.

[10] M. Loey. Arabic handwritten characters dataset. https://www.kaggle.com/mloey1/ahcd1.

[11] M. Loey. Arabic handwritten digits dataset. https://www.kaggle.com/mloey1/ahdd1.

[12] Y. Perwej and A. Chaturvedi. Neural networks for handwritten english alphabet recognition. *arXiv preprint arXiv:1205.3966*, 2012.

[13] H. Pokharna. The best explanation of convolutional neural networks on the internet!, 2016.

[14] W. Rawat and Z. Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, 29(9):2352–2449, 2017.

[15] A.-M. Saeed. Intelligent handwritten digit recognition using artificial neural network. *International Journal of Engineering Research and Applications*, 5(5):46–51, 2015.

[16] N. Sharma, T. Patnaik, and B. Kumar. Recognition for handwritten english letters: A. *International Journal of Engineering and Innovative Technology (IJEIT)*, 2(7), 2013.

[17] A. Singh, K. Bacchuwar, and A. Bhasin. A survey of ocr applications. *International Journal of Machine Learning and Computing*, 2(3):314, 2012.

[18] K. Younis. Arabic handwritten character recognition based on deep convolutional neural networks. *Jordanian Journal of Computers and Information*

Technology, 3(3):186–200, 2017.