Machine Learning Identification of Low Resolution Images of English Font Characters

Tanner Smith
smith555@rangers.uwp.edu
University of Wisconsin-Parkside
Kenosha, Wisconsin, USA

Jared Mai
mai000003@rangers.uwp.edu
University of Wisconsin-Parkside
Kenosha, Wisconsin, USA

ABSTRACT

It is not uncommon for images to be compressed to save storage or speedup transfers over the Internet. A subset of these compressed images could be screenshots of web-pages, documents, or other archival media containing text. This results in side effect of the compression/down-sampling which can cause the text within the image to become too pixelated for a human to easily make out what the original characters were. We took low resolution images of single characters from a variety of fonts and trained machine learning models to see if the models can find patterns in the image data to accurately predict the character. We used a variety of KNN, Ensemble, and Deep Learning models. The models were trained on a series of fonts with a characters of: lowercase a through z, uppercase A through Z, and digits 0 through 9. The finished models had a range of accuracy's from 70-90%. An issue with most of the models was properly predicting similar letters.

CCS CONCEPTS

• **Computing methodologies** → *Classification and regression trees*; *Neural networks*; *Image compression*; *Image manipulation*; • **Applied computing** → *Optical character recognition.*

KEYWORDS

character recognition, text, machine learning

ACM Reference Format:
Tanner Smith and Jared Mai. 2022. Machine Learning Identification of Low Resolution Images of English Font Characters. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The field of using machine learning models to identify text from image data has been around for a long time. All the way back in 1991 Bromley and Sackinger from Bell Labs used machine learning models to classify handwritten digits [2]. Today, there are tons of applications that rely on being able to identify text in an image for further processing. The google translate mobile app is one example with its feature that lets you take a picture of text to translate, instead of needing to type in in manually. In these use cases, the text is high enough quality where the human could manually read it. In this paper, we want to instead explore a sub field where the text is low quality to the point where it is difficult for a human to make out what the characters represent. Some of the potential use cases for such a machine learning model would be to potentially recover text data from documents that no longer exist except in the form of an old image/screenshot, but the image is low resolution making the text difficult to read. For example, old forums on the early internet typically deleted forum posts that were old or deemed to not be getting enough traffic to save on storage. The only remaining record of these forum posts then might be someone's screenshot. While the text on these images may be difficult to read due to pixelation from being low resolution a machine learning model could potentially find patterns in the pixel values and can quick and accurately predict the text.

2 RELATED WORKS

Much of the inspiration behind this research stemmed from Bell Labs 1991 paper by Bromley and Sackinger on a hand written digit classifier [2]. While the paper is old it laid a good foundation to base and improve this research on. Their research focused on KNN and neural networks comparing performance and how they can gain from each other.

In 2018 Zhihao Duan trained a KNN model to classify English letters A to Z and digits 0 to 9 [3]. A total of 5 different fonts were used to create the model, specific fonts used were not mentioned. All the images were generated and then reprocessed down to 10 by 10 pixels before being used to train the model. The generation process included character segmentation and size normalization. The model was then tested on 80 images of printed font characters and 20 images of handwritten characters. The model showed promising accuracy for the printed characters and poor accuracy for the handwritten characters. Daun's research was primary focused to be a preliminary investigation for to guide future research.

Another way to phrase this type of research is called scene text recognition (STR). A paper published in the International Conference on Computer Vision 2019, explored some of the common issues with current STR research [1]. They point out many issues from the training data itself, how the data is split between training and testing, and how the models are evaluated. Correctly pointed out, labeling training data for STR research is time consuming. This results in many STR researches to generate their own synthetic labeled data sets. This is the approach this paper chose to use. How the training data was generated and obtained is very important to the bias and performance of the final model. The most common approaches to generating synthetic data for STR is MJSynth (MJ) [5] and SynthText (ST)[4]. This paper's data is a simplified variation of MJ data generation. The varieties of how data is generated or aggregated for STR research creates problems directly comparing models across papers. They suggest researches indicate their training datasets clearly and only compare to models using the same training set.

The article, A Novel Deep Learning ArCAR System for Arabic Text Recognition with Character-Level Representation[6], research does similar research to this paper but unlike English characters it focuses on Arabic characters. The Arabic characters themselves are not the target but represent features. The target of the model is to classify a type of news article (Religion, Finance, Sports...). They used a convolutional neural network (CNN) with the Arabic text from the articles as inputs. The end goal of their classification is different but their methods using the binary image data as input similarly applies to this research.

3 DATA AND PRE-PROCESSING

Sourcing data for scene text recognition can be very time consuming. Researchers have to manually label all the image data with the corresponding text. While pre-labeled image datasets do exist such as **IIIT5K-Words (IIIT)**, **Street View Text (SVT)**, and **ICDAR2003 (IC03)**, the image data from these datasets would have still required heavy modification for the planned models for this research [1]. Therefore we opted to generate a synthetic data set to get tons of labeled image data quickly. Because fonts are rendered consistently across machines the synthetically generated images would most likely yield similar performance if documents were manually segmented and labeled .

3.1 Image Generation

Images were generated using Python Image Library **PIL/PILLOW**. First a base all white image of 64 by 64 pixels, with one byte per pixel. Then a font is selected and font size is set to 60. With the selected a font a character is rendered in the middle of the base white image. For each font the process is repeated for all lower case characters a to z, all uppercase characters A to Z, and all digits 0 9. A total 62 images were generated for each font. All font files that came with a default install of Windows 11 Pro were used to generate images. However, only common fonts such as Times New Roman, Arial, or similarly looking fonts to those were used in final model training.

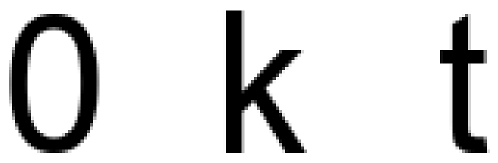


Figure 1: Example Images of characters (0, k, t) | Font = Arial

3.2 Down sampling

The images above are clear and easy for a human to identify, we want to train a model on hard to read low resolution text therefore we need to down sample the base images. Using the python OpenCV **(Cv2)** library from google the base images were down sampled from 64 by 64 pixels to 8 by 8 pixels. OpenCV provides several interpolation techniques[8]. An ensemble of three interpolation techniques were chosen, Inter Area, Inter Linear, and Inter Cubic. Therefore each base image is converted into three separate down scaled versions. The figures below show an example of the 3 seperate types of downsampling methods

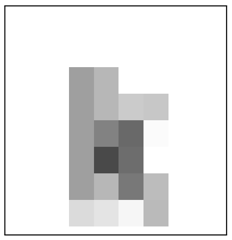


Figure 2: Font = Arial | Interpolation = Inter Area | Char = k

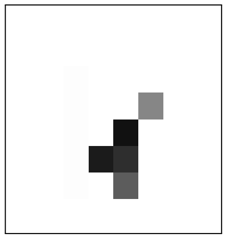


Figure 3: Font = Arial | Interpolation = Inter Linear| Char = k

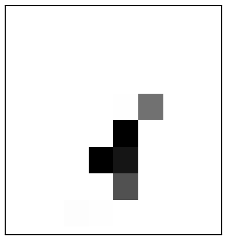


Figure 4: Font = Arial | Interpolation = Inter Cubic | Char = k

3.3 Final Data Set

After all the generation and down sampling the final data is comprised of 21 different fonts with a total of 4464, 8x8, black and white 8 bit per pixel images. The file name of each image contains: Font, Font Size, Original Size, Current Size, Character. These images were then loaded into a Python Pandas data frame for model training.

4 KNN MODELING

K-Nearest Neighbors is an artificial intelligence prediction model that classifies data based off how how close it is to k neighbors. The distance one data object is from another depends on the features and distance metric used. In this paper the features used in distance calculations are each pixel value (0-255) unsigned integers that comprise the image.

4.1 Distance Metrics

Some common distance metrics used for KNN are Euclidean, Manhattan, and Minkowski.

Euclidean

$$\sqrt{\sum_{i=1}^{m}(x_i - y_i)^2}$$

Manhattan

$$\sum_{i=1}^{m}|x_i - y_i|$$

Minkowski

$$(\sum_{i=1}^{m}|x_i - y_i|^r)^{\frac{1}{r}}$$

Using a different distance metric can yield different results [7]. For the experimental results we tested the 3 distance metrics and found that they did not yield a significant difference on initial testing. Going forward we used the default distance metric from the Sklearn python package "Minkowski" to build all the models.

4.2 Initial KNN Models

To get a better idea on how we can improve the models we constructed some basic KNN models. A model was generated for the entire data set, and three more subsets of the three down sampling techniques. A completely random train test split of 80% to 20% with a default k value of 5. These models were the simplest and quickest ones we were able to build. We took this data as the baseline to beat when we try to improve the model.

Table 1: Initial KNN Model Avg Accuracy's

Model	Accuracy
All	~70%
Inter Area	~75%
Inter Linear	~50%
Inter Cubic	~50%

4.3 Fixing the train test split

While with the train test split method we used for the initial models showed some promising results we spotted some major potential flaws right away. The train test split we were using from the python Sklearn library splits the data treating all data points equally. However, with the nature of our data set splitting data *completely randomly* can lead to some problems. An example of this issue could be: during one sample of the training set it randomly selects the letter W from the font list and a much higher rate to the letter E. This would result in the training set to have a ton of the letter E and few letter W. While the testing set has more of the letter W and less of the letter E. This may lead the model having an unequal bias to with more training on one letter over the other. To fix these we implemented a method that splits the data into training testing sets of an entire font. This new train test split still allows us to split the data roughly into an 80:20 ratio of training and testing data, and ensures that each letter has an equal distribution in the training and testing data.

4.4 New Train Test Split KKN models

With the new train test split method we repeated the initial tests using the new train test split. While knowing that the new train test split more evenly splits the data we were unsure if it would improve the model accuracy.

Table 2: New Train Test Split KNN Model Avg Accuracy's

Model	Accuracy
All	~70%
Inter Area	~80%
Inter Linear	~55%
Inter Cubic	~55%

Looking at the data in Table 2 we saw that on the model trained only using an individual down sampling method they preformed roughly 5% better than the models that used the original train test split. We have more accurate models and models that represent the problem better with evenly split classes. However, occasionally we would see a model with a very poor accuracy. These poor accuracy models were rare and an outlier. What it likely implies is that there are some specific font sets in the data set that are more different than the others which makes it harder for the model to predict them.

4.5 Finding the optimal K value

One last thing we tried in order to improve the performance of the KNN models was tweaking the K value. The K value is the amount of nearest neighbors the model will use when predicting a new data point. We took the best performing model so far, which was the Inter Area model using the new train test split, and tested it with K values from 1 to 20 multiple times. We then plotted the accuracy as a function of K. Overall we saw a downward trend as the K value increased. Figure 5 shows some sample charts with the observed downward trend. Occasionally we did not see this trend and the accuracy seemed to bounce around sporadically. This was accompanied with an overall low accuracy which implies that the fonts uses for testing were significantly different than the training fonts. These sporadic trends were not included in figures.

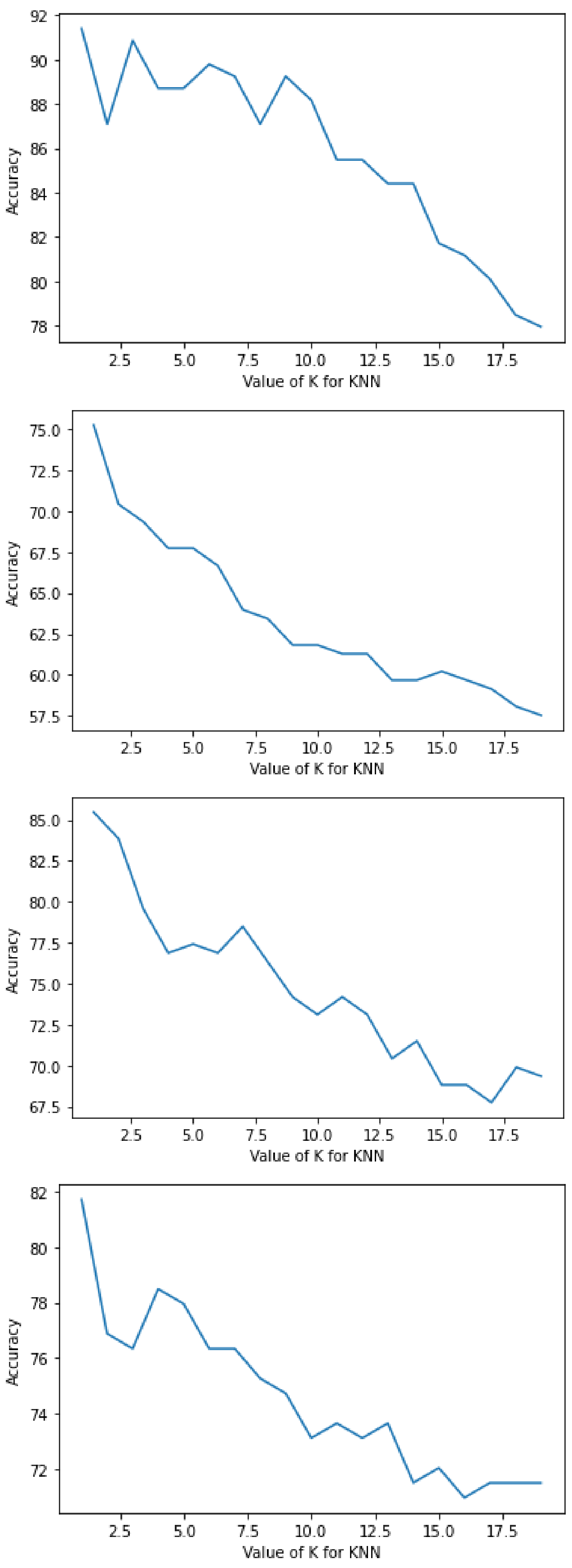


Figure 5: K Value vs Accuracy for many InterArea Splits

4.6 KNN Modeling Conclusion

Overall using KNN to predict the letters seemed pretty promising. Predicting text at such a low resolution is a hard task and getting an accuracy of 70 to 80 percent is promising.

5 ENSEMBLE LEARNING

Another type of machine learning model we tried was Ensemble Learning. In general ensemble models use several smaller simple models to come together to create one overall model. How the smaller models are build depends on the ensemble modeling technique. Two main ensemble modeling techniques are boosting and bagging. We chose to use a boosting model. In boosting each sub model is trained sequentially instead of in parallel. The error of each previous sub model is used to tweak the weights of input features for the next model with the attempt to lower error. In the end there are tons of sub models that increasingly learned from each other to come together into one main model.

5.1 Ensemble Learning Results

We trained a gradient boosting classifier from the python Sklearn on our data set. Due to the nature of boosting the model took 30 minutes to complete only using a subset of the entire data set. This subset still used all down sampling types. The long training time did pay off because the model has a **93%** accuracy. This is much higher than the previous KNN best model and it uses all the down sampling types. A potential reason as to why this model performed so well is that it figured out the weights of most important and least important pixels. Not all pixels of the characters are important as each other. For example the outside pixels are almost always white, while the pixels near the middle vary much more. Figure 6 shows a potential heat map example of how the pixels may be weighted in terms of importance.

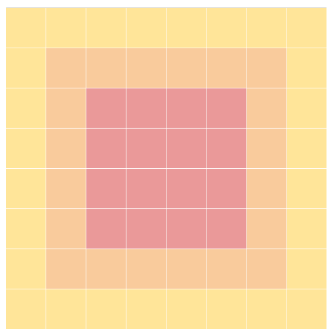
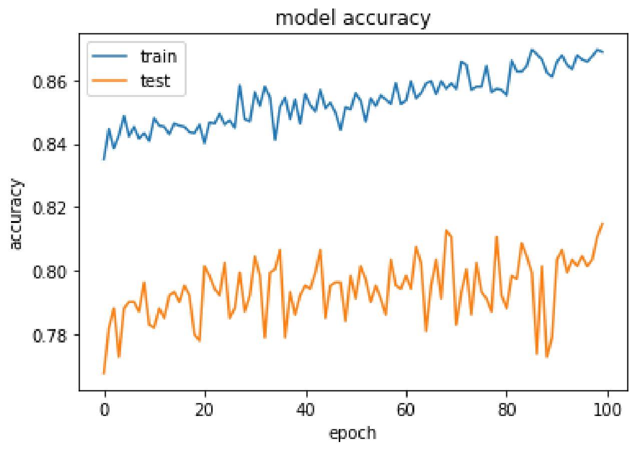


Figure 6: Theoretical Pixel Weight Heat Map

6 DEEP LEARNING

Deep learning is a very popular machine learning technique in image recognition. Several of the related papers to ours used a form of a deep learning model [2, 5]. We trained a simple shallow multi-layer perception (MLP) deep learning model on our data set. Our MLP model uses four dense layers of sizes (64, 32, 16, 60). The model was trained for 100 epochs. Figure 7 shows the loss and accuracy per epoch.



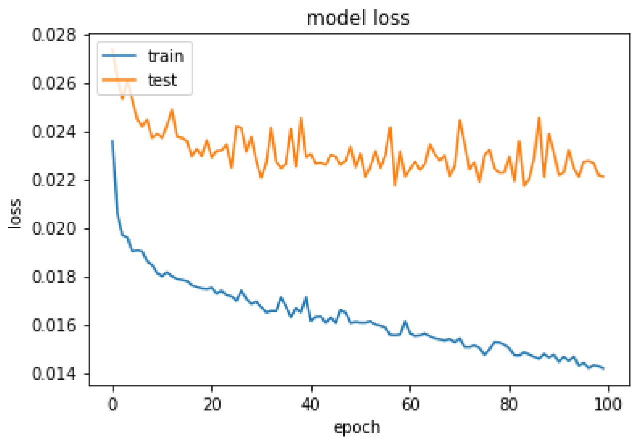


Figure 7: Deep Learning Model Loss & Accuracy per Epoch

6.1 Deep Learning Results

The deep learning model preformed well but we expected it to do better. The lower than expected performance of the model could be due to the fact that our image resolution (i.e feature count) is relatively low with 64 features. Deep learning uses tensors which are often speed up in compute time with graphical processing units (GPU's) which can do parallel calculations on an entire tensor at once. The size of our features for this machine learning problem does not very mix well with deep learning. Other studies typical use larger image sizes when working with deep learning and see a higher accuracy[5].

7 MODEL COMPARISON & DISCUSSION

With several models varying in performance we can compare, rank, and contrast them with each other to try and find what model is most optimal for this type of problem. Then we can figure out where the models could potentially be further improved. In terms of ranks the ensemble model (gradient boosting) came out on top, followed by the deep learning model, and finally in last was the best KNN model. It is disappointed to see the KNN model at the bottom since we tried several methods to improve it's accuracy. On the other hand we did not try to increase the ensemble boosting classifier or the deep learning MLP model, but they performed better than the best KNN model. Our ensemble model was put together very quickly and was not initially planned at the start of this research as the related research papers we looked into did not use this type of model. We also do think an accuracy around 95% might be the upper ceiling as some of the down sampled text turned into all white pixels except for 2 to 3 non white pixels, making it extremely hard to extract meaning data and features from the image. Therefore all of those images we expect to see incorrectly predicted most of the time. The images like that were mostly the skinny tall digits and characters such as (1, i l, t...). These were the most common mistakes between all models which is why we think the upper limit is around 95%.

8 CONCLUSION

Predicting low resolution text from image data is a challenge that can have some very practical use cases retrieving data from compressed images. We found that a variety of machine learning models can predict the individual characters with promising accuracy. We also identified the difficult challenge of predicting similar letters that lose their distinct characteristics at a low resolution. By splitting data between whole sets of fonts we fixed the class distribution but also found that a font could greatly affect the models results. Similar looking fonts would perform better and when we chose fonts we focused on trying to pick fonts that looked similar. The new data split showed that even though they looked similar some are still significantly different. Obtaining a 100% accurate model that can predict low resolution letters is likely impossible. Our models range of 70 to 90 accuracy is a good preliminary step.

9 FUTURE WORK

There are many things that we think could expand upon the results of this study. In the real world when letters get down scaled they are often next to other letter pertaining to the world they are apart of. The neighboring letters may influence the border pixel values of an isolated letter changing how a model would interpret it. A more robust model than ours would take isolated letters from images that were down sampled of entire text documents. Another thing that could further expand upon this work would be to test several decreasing down sampled sizes. Using the models from this study as a reference we would build the same models but using images of increasing or decreasing resolution to find the sweet spot that a model needs in terms of pixels to be somewhat accurate. 8 by 8 was an arbitrary low value we chose based off other studies that attempted to classify characters. Lastly, in order to solve the problem of similar looking letters that are just too difficult to accurately predict like (O and 0). A secondary natural language processing model (NLP) could be trained to take in misspelled words and find possible letters that fill in the blanks to make a real word. Then the models we built that predicts individual characters could be used to predict a word one letter at a time and any. Then the predicted set of letters can be fed into the NLP model to correct possible mistakes. For example lets say our model predicted m0p with a zero instead of an 'o'. The NLP model will then recognize the mistake and attempt to replace the zero with the correct 'o'.

ACKNOWLEDGMENTS

Thank you to Dr. Vijayalakshmi Ramasamy, from the University Of Wisconsin Parkside for guiding us through this research.

REFERENCES

[1] Jeonghun Baek, Geewook Kim, Junyeop Lee, Sungrae Park, Dongyoon Han, Sangdoo Yun, Seong Joon Oh, and Hwalsuk Lee. 2019. What Is Wrong With Scene Text Recognition Model Comparisons? Dataset and Model Analysis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

[2] J. Bromley and E. Sackinger. 1991. Neural-Network and k-Nearest-neighbor Classifiers. , 11359–910819 pages. http://oro.open.ac.uk/35666/

[3] Zhihao Duan. 2018. Characters Recognition of Binary Image Using KNN. In *Proceedings of the 4th International Conference on Virtual Reality* (Hong Kong, Hong Kong) *(ICVR 2018)*. Association for Computing Machinery, New York, NY, USA, 116–118. https://doi.org/10.1145/3198910.3234651

[4] Ankush Gupta, Andrea Vedaldi, and Andrew Zisserman. 2016. Synthetic Data for Text Localisation in Natural Images. *CoRR* abs/1604.06646 (2016). arXiv:1604.06646 http://arxiv.org/abs/1604.06646

[5] Max Jaderberg, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2014. Synthetic Data and Artificial Neural Networks for Natural Scene Text Recognition.

[6] Abdullah Y. Muaad, Mugahed A. Al-antari, Sungyoung Lee, and Hanumanthappa Jayappa Davanagere. 2022. A Novel Deep Learning ArCAR System for Arabic Text Recognition with Character-Level Representation. *Computer Sciences amp; Mathematics Forum* 2, 1 (2022). https://doi.org/10.3390/IOCA2021-10903

[7] Archana Singh, Avantika Yadav, and Ajay Rana. 2013. K-means with three different distance metrics. *International Journal of Computer Applications* 67, 10 (2013).

[8] Anthony Tanbakuchi. 2020. Comparison of openCV Interpolation methods by Anthony Tanbakuchi. https://gist.github.com/georgeblck/e3e0274d725c858ba98b1c36c14e2835

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009