Module 8: Portfolio Project

AI Use – Case Problem With Solution - Paper

Nolan Byrnes

CSC510 – Foundations of Artificial Intelligence

Colorado State University – Global Campus

Professor Bingdong Li

July 7th, 2023

Module 8: Portfolio Project

One of the challenges with scanning or taking a picture of a document is that the characters within the image are not digitized. This is because a computer image is “represented in the form of a matrix where each element consists of single pixels” (Educative IO, n.d., para. 2). To digitize the text, we must first preprocess the matrix of pixel values by identifying and extracting the characters from the image and then characterizing each of the extracted characters to its printed value. To find the relevant data within the image, I used Computer Vision techniques to preprocess images and detect patterns of printed content in an image. I then sent the filtered data through an AI model that I trained by using supervised learning to predict what each character represented. The final output of the program is the digitized text that was extracted. This paper goes over how I trained a model to categorize characters that are fed it, how I used computer vision to extract characters from images, and how I used the trained model to deliver the final output, which was the digitized text extracted out of the image.

**Preprocessing Training Dataset**

Peterson explains that “supervised learning is an approach to creating artificial intelligence (AI), where a computer algorithm is trained on input data that has been labeled for a particular output” (Peterson, para. 1). To teach an AI Model how to differentiate between different characters, we would show it a large dataset of labelled characters, and have it study the differences until it is able to categorize characters accurately. The data set which I used contained 745,000 images of characters from 153 different character fonts, which were obtained by cameras and scanners (Lyman, 2016). Mitsa mentions that the rule of thumb for image classification is to have “1,000 images per class”, so with the data set that I am using, I should have enough data to train an effective model (Mitsa, 2019, para. 10).

For the scope of this project, I focused only on the Unicode characters from 65 through 128, which included some special characters, and the alphabet capitalized and uncapitalized, for a total of 58 categories (Vertex42, n.d., table 1). I had to subtract 65, so that the classes were from 0-58, and then when printing the characters, I had to add 65 to the final classification so that when I used the chr() function, the correct character was being printed. The original data set also had the character images in greyscale, with the pixel values being from 0-255, however, I converted the images to a binary format to make it easier for the computer vision portion of the project to match the data that the model would be trained on. When prepping the dataset to be used for training, I separated the original height, original width, and the 400 pixel values from the unicode value of the character, with x\_train, and x\_test being the character data, and y\_train and y\_test being the expected values. for the test data set, I took the same number of characters from each font from the data set, to ensure all the different fonts were being tested. Lastly, I had all the training data shuffled, to make sure that the data’s fonts and characters were not in any order. After I had the training dataset pre-processed, I had my program train on the training data set (x\_train and y\_train).

**Training the Convolutional Neural Network**

After obtaining and preprocessing the data I was going to use to train my model, I fed the training data to a Convolutional Neural Network (CNN) to learn how to differentiate each of the characters that were being fed to it. Kumar explains that “Convolutional Neural Networks (CNNs) are deep neural networks that have the capability to classify and segment images” which can be trained with supervised machine learning (Kumar, 2023, para. 1). To create the model, I used Keras, which is a high-level API which “simplifies the implementation of complex neural networks with its easy-to-use framework” (ActiveState, 2022, para. 2). The Keras Sequential model I used contained a dense layer which utilized the relu activation function, a dropout layer, and another dense layer with the softmax activation function. Overfitting is where a “statistical model fits exactly against it’s training data” which makes it so where it “cannot perform accurately against unseen data” (IBM, n.d., para. 1). To prevent overfitting from occurring, I used a dropout layer which “randomly sets input units to 0” to prevent overfitting and it also helps to adjust for any noise that might exist in the inputs when predicting (Keras Team, n.d.-a, para. 2). For the amount of epochs I trained the model on, I used Keras Early Stopping callback which will “stop training when a monitored metric has stopped improving” (Keras Team, n.d.-b, para. 1). By using this callback, I could have the model train until it was receiving the best accuracy, which reduced the guesswork I had to do when determining the optimal number of epochs that the model should train on.

After the model was trained, I had it test its categorization abilities against the test data set, which is data that the model has not seen before during the training, to emulate what it would be receiving when it is being used in practice, and it characterized 80% of the test characters I fed it correctly. I found that when I normalized the pixel values to be either zero or one, where I had the zero-value representing black pixels, and the one value representing white pixels, my trained model returned better results. Keldenich explains that normalizing data places the values into the same range as our activation functions, which “allows for less frequent non-zero gradients during training, and therefore the neurons in our network will learn faster” (Keldenich, 2021, para. 30).

**Preprocessing Image Data with Computer Vision**

After the model was trained, I was ready to feed it the characters to be categorized, but first, I needed to preprocess the input image data, extracting the characters to be sent through the model. For the computer to ‘read’ and preprocess the images so that I can extract the characters, I used an open-sourced computer vision software library called OpenCV, “which was built to provide a common infrastructure for computer vision applications” (OpenCV, 2020, para. 1). Sharda mentions that there are four main steps of data preprocessing which include data consolidation, cleaning, transformation , and reduction (Sharda, 2019, pg. 129). First, I had the program reduce noise and make it easier to search through the image by converting the image into greyscale, then into binary.

This is so that the pixel values could only have one of two values, rather than having characters within the range of 0-255. I consolidated the data by first searching for lines of text, and then for each line identified, I would look for the contours within each line. This helped the program focus on one line of text at a time, extracting the characters from left to right, to ensure the characters being extracted were in the correct order. I transformed the data by adding the original height and width of each of the detected characters identified, resized the Region of Interest (ROI) to be a 20 by 20-pixel image. Finally, I reduced the data by flattening out the dimensions, to where all pixel values were in a 2d array along with the original height and width as the first and second index, and the rest of the pixel values from index 3 - 402.

For each of the areas that represented the lines of text, I used OpenCV’s findContours() function which identifies all the contours within the given area. In this context, each contour would represent a character. I then had the contours sorted by the x values, so that all the identified characters within the line would be in order from left to right. For each of the characters, I resized it to a 20 by 20-pixel array to match the data that the model was trained on and included the original height and width before the ROI was resized to a 20 by 20-pixel matrix.

One of the challenges that I encountered when extracting the text was detecting a white space character. This is because typically, a white space character is just white space, so detecting them by looking for contours would not happen. To overcome this challenge, I used symbolic representation to determine if a white space existed between characters. I defined the existence of a white space in between characters by getting the average distance between characters, if between characters had a distance between them that was an outlier (150% over the average), it would represent a white space, and allowed the program to separate all the characters on each line into groups of words.

By using symbolic representations for separate lines and white space characters, I was able to organize the data that was being sent to the AI model to where each character being extracted would be grouped with its word, and the words would be grouped together based on the line that they were on. When I had the model predict the character values, it was able to organize the predicted character values that would give us separate words and lines, rather than grouping all characters together as one word.

**Implementation**

After the input images data from the inputs folder was extracted and packaged up to where all characters were grouped into words, and words being grouped into lines, I had the trained model predict what each of the characters represented. The trained model would give back its predictions on which Unicode character the data represented. I used the argmax() function to obtain the Unicode character that the trained model felt the data represented with the most confidence, and then converted the Unicode character to its string character representation to build the resulting string of characters extracted from the input image.

**Conclusion**

The problem I was solving with my portfolio project was to have the ability to digitize text off images. I was able to solve this problem by training a CNN to recognize patterns of all the alphabet characters a-z capitalized and uncapitalized. I preprocessed the data set I acquired to make it easier for my CNN to train on. After training the CNN, I used computer vision to extract characters from an image, preprocess it so that the data extracted matches the same as what the CNN was trained on, which was having the original height and width of the extracted characters, and the resized characters to fit a 20 by 20 matrix of pixel values flattened. I introduced symbolic planning so that the characters that the CNN model predicted would be organized into words and lines.

The program represented knowledge by using a logical approach with computer vision to identify patterns of pixels which represent characters and utilizing a neural network to categorize the extracted characters from a given image. As a result, the program was able to successfully categorize all the characters that were extracted using computer vision and fed through the CNN, outputting the digitized text that was being displayed on the image.

**REFERENCES**

ActiveState. (2022, July 11). What is a Keras model and how to use it to make predictions. <https://www.activestate.com/resources/quick-reads/what-is-a-keras-model/>

Educative IO. (n.d.). What are digital images? Educative: Interactive Courses for Software Developers. <https://www.educative.io/answers/what-are-digital-images>

IBM. (n.d.). What is Overfitting? IBM - United States. <https://www.ibm.com/topics/overfitting>

Keldenich, T. (2021, July 13). Why and how to normalize data for computer vision (with PyTorch). Inside Machine Learning. <https://inside-machinelearning.com/en/why-and-how-to-normalize-data-object-detection-on-image-in-pytorch-part-1/>

Keras Team. (n.d.-a). Keras documentation: Dropout layer. Keras: Deep Learning for humans. <https://keras.io/api/layers/regularization_layers/dropout/>

Keras Team. (n.d.-b). Keras documentation: EarlyStopping. Keras: Deep Learning for humans. <https://keras.io/api/callbacks/early_stopping/>

Lyman,Richard. (2016). Character Font Images. UCI Machine Learning Repository. <https://doi.org/10.24432/C5X61Q>.

Mitsa, T. (2019, April 23). How do you know you have enough training data? Medium. <https://towardsdatascience.com/how-do-you-know-you-have-enough-training-data-ad9b1fd679ee>

Petersson, D. (2021, March 26). What is supervised learning? SearchEnterpriseAI. <https://www.techtarget.com/searchenterpriseai/definition/supervised-learning>

OpenCV. (2020, November 4). About. <https://opencv.org/about/>

Sharda, R. D., & Delen, D. (2019). Analytics, Data Science, & Artificial Intelligence (11th ed.). Pearson.

Vertex42. (n.d.). Huge list of Unicode character symbols. Vertex42.com. <https://www.vertex42.com/ExcelTips/unicode-symbols.html#utf8>