RGB Image Character Classification

Gihwan "Kris" Kwon[1]

University of Washington Bothell, Bothell WA 98011, USA
kwonerstone3@gmail.com

Abstract. This is Abstract.

Keywords: First keyword · second key word.

1 Introduction

Meme is a humorous image, video, or piece of text that is copied and spread rapidly by Internet users. Sometimes, people want to search meme online and show it to their friends that they thought it was funny. However, often, it is hard to find the meme that a person is looking for. Currently, online meme search engines are relying on the hash tags to match user inputs with the results. However, normally, users do not catch the phrase from the hash tags, but the actual contents in the image. Therefore, it would be useful to have meme search engines to have an algorithm such that can find the memes based on the word contents inside the images.

Convolutional Neural Networks is a popular method to recognize images and classify its label. Currently, there are many research focusing on utilizing this method for recognizing text images in real life. However, classifying memes is different in that words in the memes are graphically added as contents. Therefore, making a model for classifying graphically oriented characters would be useful for developing a search engines for memes.

The main goal of this project is to contribute to the developers who would build a meme search engine for the users. To begin with, this project will focus on classifying graphical characters. This paper will explain how the model was built with the convolutional neural networking methods.

2 Method

The method used for this project is the Convolutional Neural Network. This section explains methods for this project: Data preprocessing, Training, Prediction.

2.1 Data Preprocessing

The original data sets are organized in the .csv file. The objects are consist of attributes such as *font* (string), *fontVariant* (string), *label*(int), *strength* (int), *italic* (int), *orientation* (int), *m top* (int), *m left* (int), *originalH* (int), *originalW* (int), *h* (int), *w* (int), and 20 by 20 size grey scale image matrix. Most of the attribute names are self-explanatory. However, to be more specific, *m top* and *m left* are coordiate where image of the character starts in the real image in terms of row and column. Moreover, *originalH* and *originalW* are original image size. Finally, *h* and texitw are the height and weidth of the image.

To begin with, some of the attributes are omitted in this process because recognizing font and whether the character is italic are not the part of this project. Moreover, all the character images are original image. In other words, image in the datasets only contains one character. Therefore, *label* of the character and the 20 by 20 size grey scale image is going to be used in this project.

Next step is to convert the grey scale matrix into the RGB matrix which is three dimensional. The result matrix after preprocessing would produce 20 by 20 by 3 image matrix. To be specific, conversion took place by appending the original grey scale matrix into the result matrix three times. The result is the N number of RGB image matrices.

Furthermore, data preprocessing step produces a vector that values in each index contains character label and this corresponds the image matrices stored in the image matrices. Finally, this step produces a matrix size of N by the number of unique characters. The value in the indices are initially zeros in floating point. After image conversions, based on the image label, vector for each image will contain 1.0 where index corresponds the image label. For instance, if there are five different unique image labels in 10 different objects, the result will be ten by five matrix containing 1.0 in the vector where index matches the character label, otherwise values will be 0.0s. In this case, the floating point one value specifies that the index, character label, is true for corresponding image.

2.2 Training

The method of Convolutional Nerual Networks requires multiple layers in the process of trianing. In a big picture, the layers could be divided into three layers: input, hidden, and output layers. The input layer receives input, data such as matrices, and output layer produces the result values produced by the model. Result values are N by the number of unique characters matrix that each vector contains the probability of input to be the character of the corresponding index numbers. The layers are implemented while utilizing the Tensorflow python package.

The hidden are further divided into three layers: Convolution, Flattening,

and Fully Connected layers. For this project, there are three convolution layers, one flattening layer, and two fully connected layers.

Convolution Layer is filtering layer. Convolution layer applies filter to find patterns in the input image. The size of the filter is three by three matrices. The values in the matrices are initially random numbers, which are weights. Also, biases are initially randomized. However, this values will be updated in the back propagation stage.

The number of filters are 32 in first two convolution layers and 64 in the last layer. Then, the filters with weights will be applied to the input image and biases will be added after convolution phase. After filtering the image and added biases, this layer will pool the max values in each filters. Finally, outputs from the pooling function will be fed to the Relu function.

To be more specific, filtering a RGB image, size of 20 by 20 by three, with three by three filter and max pooling function will create a matrix size of 18 by 18 by three. The matrix will be padded with zeros to reserve the original input size. This output will be fed into the Relu function to remove negative values.

Flattening Layer is reshaping outputs from convolution layers before feeding into fully connected layer. Simply, this layer "use the reshape operation to create a single dimensional tensor."[1]

Fully Connected Layer is defined to "receive input from all the neurons in the previous layer"[1] by applying weights and biases to the input vectors. Initially, weights and biases are randomly chosen. However, this will be adjusted during the back propagation process. After matrix multiplication, again, the reulu function will be applied to remove negative values. Then the result outputs will be the probability of each tensor being a prediction.

After inputs going through the layers, the model optimize the randomly initialized weights and biases. This process is called **back propagation**. From the output tensor from the last fully connected layer, this model utilizez the soft max function to find the highest probabilities of the result. Then, it compares the predicted labels to the actual character label. Based on the result of the prediction, the model calcualtes the cost by utilizing the cross entropy function and the average of its result. Finally, this model "uses AdamOptimizer for gradient calculation and weight optimization."[1] This model tries this method to minimize the cost with a learning rate of 0.0001.

3 Experiment and Result

This section will explain how the model was trained, validated, and tested. Several adjustments were made to improve the model's prediction accuracy. For the

training process, the model was trained with training data sets and validated with validation data sets. On the other hand, testing data sets are completely different from the training and validation data sets. The number of training data sets are 47930 and validation sets are 5325. The validation set size is roughly 10 percent of the training data sets. Five different experiment took place to improve the prediction accuracy. In order to see the difference in the experiment, the model was trained with two different learning rate and the number of epochs; Learning rate was 0.0001 and 0.00001; The number of epochs are 17, 21, and 27. Finally, one batch of data sets contain 32 data sets.

To begin with, during the training process, the goal is to minimize the validation loss. Therefore, training with different independent variables such as epoch sizes and learning rates produced the various accuracy when predicting. The training processes and the results of prediction with the prediction accuracy are provided in the figures below.

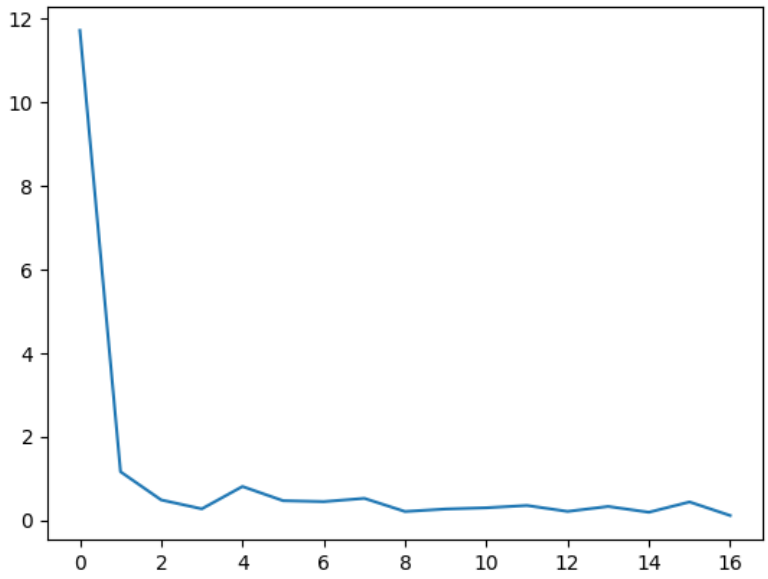

Fig. 1. Validation loss graph for epoch size 17 with learning rate 0.0001.

Above figures are the output logs for learning rate of 0.0001 with different number of iterations. The first training seems

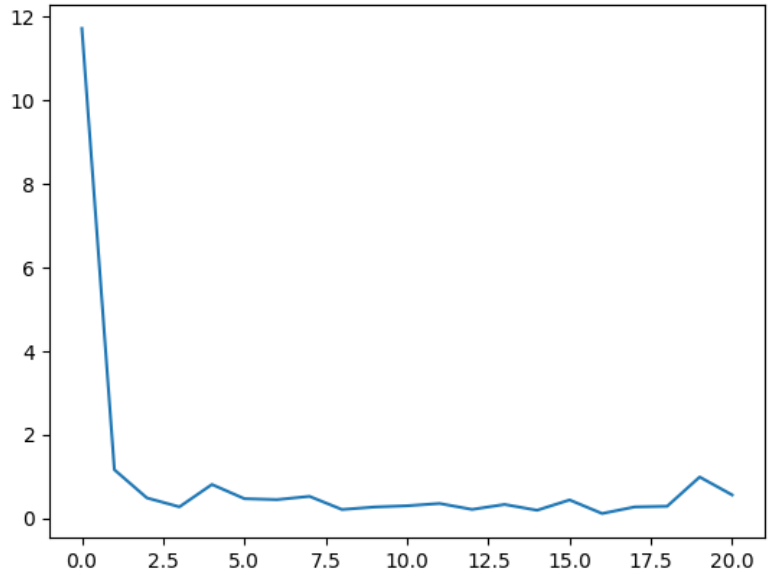

Fig. 2. Validation loss graph for epoch size 21 with learning rate 0.0001.

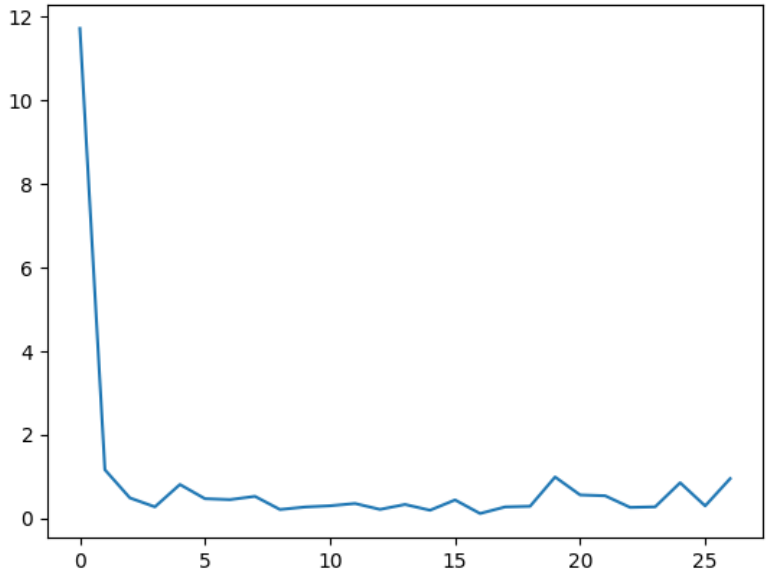

Fig. 3. Validation loss graph for epoch size 27 with learning rate 0.0001.

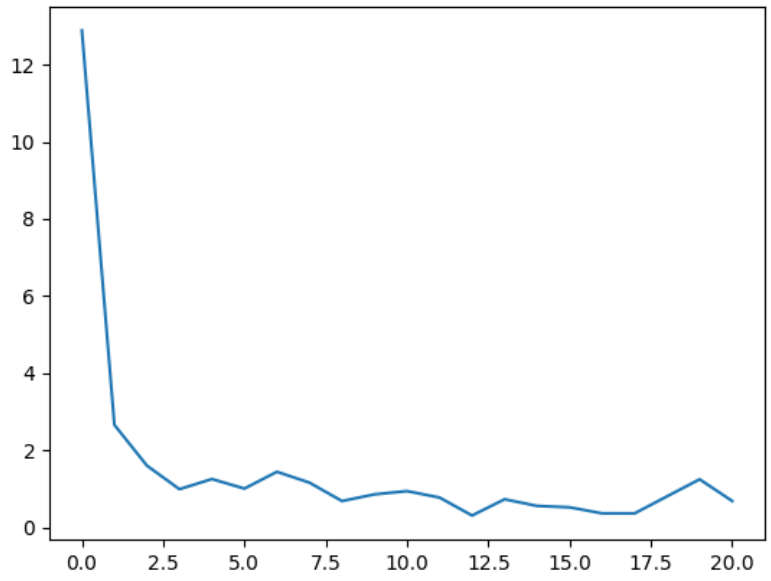

Fig. 4. Validation loss graph for epoch size 21 with learning rate 0.00001.

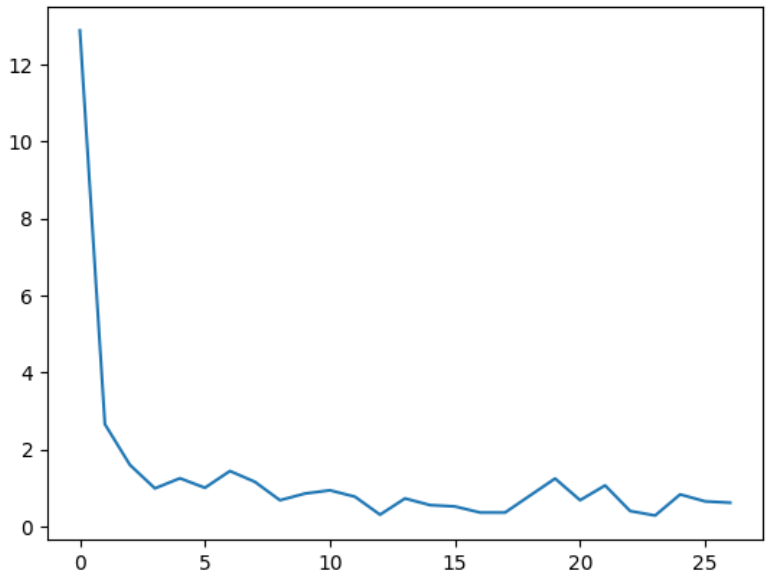

Fig. 5. Validation loss graph for epoch size 27 with learning rate 0.00001.

3.1 Training

The number of training data sets are 47930 and validation sets are 5325. The validation set size is roughly 10 percent of the training data sets. Five different experiment took place to improve the prediction accuracy.

3.2 Prediction

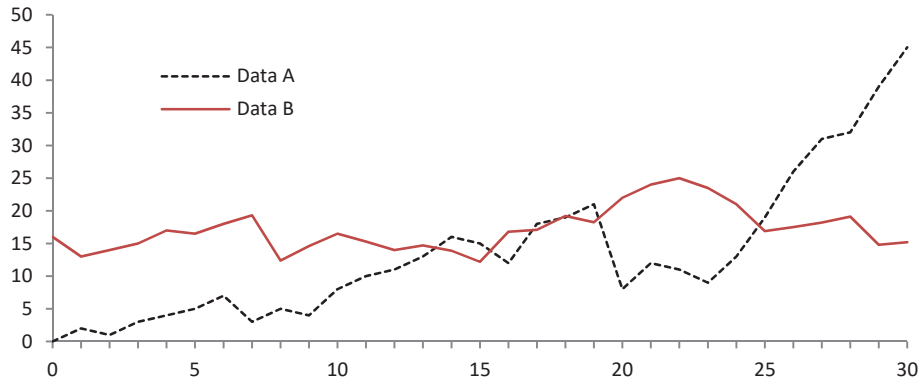

Fig. 6. A figure caption is always placed below the illustration. Please note that short captions are centered, while long ones are justified by the macro package automatically.

Hello World![1]

$$f(x) = x^2 \tag{1}$$

4 Conclusion

References

1. SACHAN, A.: Tensorflow tutorial 2: image classifier using convolutional neural network (Feb 2017), https://cv-tricks.com/tensorflow-tutorial/training-convolutional-neural-network-for-image-classification/