Converting graphs to ASCII art with convolutional neural network

Jiongyi Li

Department of Engineering, Pennsylvania State University, State College, PA 16802, USA

jzl6831@psu.edu

Abstract. In an era marked by data's rapid proliferation, novel data representation and analysis methods have become increasingly significant. However, translating complex graphical structures into character-based representations remains a largely unexplored territory. This problem holds considerable importance due to its potential applications in fields like data compression and the development of innovative graphical interfaces. This study seeks to address this gap by proposing a unique methodology that uses a Convolutional Neural Network (CNN) model to translate graphical images into corresponding character arrangements. The approach involves preprocessing graphical inputs using edge detection techniques, slicing the pre-processed graph into specific columns, and feeding the resulting slices into the trained Convolutional Neural Network (CNN) model for character prediction. I interpret the SoftMax output of the model to determine the most probable character for each slice. The results indicate that the granularity of slicing impacts the accuracy of the generated character-laden graph, with higher granularity producing more precise translations. This finding demonstrates the model's ability to effectively translate graphical data into character-based representations, offering promising prospects for future study in this domain.

Keywords: convolutional neural network (CNN), graph-to-character translation, image preprocessing.

1. Introduction

The progression of machine learning and artificial intelligence technologies has sparked novel applications [1]. Image transformation and representation is one area of focus, affecting digital media, data transmission, and accessibility technologies [2]. Two critical research areas include image compression and style transfer, each offering unique challenges and objectives [3].

Image compression aims to reduce the storage size of an image without significantly compromising its quality [5]. Efficient storage and transmission of images have become increasingly important in the era of big data [5]. Numerous studies have utilized machine learning techniques to improve image compression. While some specific methods include approaches such as the discrete wavelet transform or the discrete cosine transform, others have considered the application of genetic algorithms for optimization, all showing promising outcomes [3].

On the other hand, image style transfer looks to alter an image's style while maintaining its content [7]. This concept finds application in digital art creation, photo editing, and augmented reality [8].

Groundbreaking work by Gatys et al. introduced the idea of dissociating and recombining image content and style using convolutional neural networks, which initiated a revolution in style transfer research [9]. Further studies, such as "Perceptual Losses for Real-Time Style Transfer and Super-Resolution" by Johnson et al. and "Instance Normalization: The Missing Ingredient for Fast Stylization" by Ulyanov et al., have refined and enhanced these techniques [9].

This project explores a novel application of these concepts – transforming graphical images into a character-based representation using a Convolutional Neural Network (CNN) [3]. This study aims to blend the principles of image compression and style transfer to create a new form of image representation, broadening the possibilities for data transmission and accessibility [11].

2. Method

2.1. Pipeline

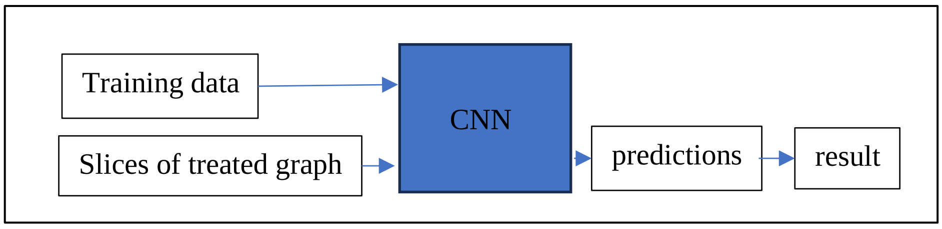

Figure 1. The general structure of the system.

The pipeline for my process is visually represented in the graph (Figure 1.) provided on the place above this paragraph, and it generally involves two main stages: the training of the Convolutional Neural Network (CNN) using my prepared dataset and the use of this trained model to generate character representations of graphical data.

In the first stage of the pipeline, I employ my prepared dataset - a collection of diverse font styles rendered as images and processed using Canny edge detection - to train my Convolutional Neural Network (CNN). The architecture of my Convolutional Neural Network (CNN) is based on the Inception V3 model, a popular choice in image classification due to its remarkable performance and efficient use of computational resources.

This training process teaches Convolutional Neural Network (CNN) to recognize and distinguish between different characters. The model learns the unique edge patterns that correspond to each character in my 95-class set, effectively gaining the ability to differentiate and predict them based on input image data.

Once the model is satisfactorily trained, I progress to the second stage of the pipeline. Here, I feed my treated graphical data into the trained Convolutional Neural Network (CNN). Each slice of this input graph, processed similarly to the training data, represents a block of potential character space. The Convolutional Neural Network (CNN) then interprets these slices, predicting the most suitable character representation based on its learned associations from the training phase.

Through these stages, the pipeline successfully translates graphical data into a series of predicted characters, demonstrating my trained Convolutional Neural Network (CNN) model's powerful capabilities in image-to-character conversion.

2.2. Convolutional neural network (CNN)

For constructing my model, I leveraged the established architecture of the Inception V3 model, a pre-existing solution recognized for its proficiency in handling image classification tasks. By using this model as my starting point, I was able to take advantage of its complex architecture, which employs numerous filter sizes to extract different degrees of abstraction from the input data and, eventually, produce more accurate predictions.

Upon this foundation, I tailored a Convolutional Neural Network (CNN) designed to handle my unique task of transforming graphic inputs into character predictions. This involved configuring the model to accept input data that matches the shape of my prepared training dataset and output data that aligns with the number of distinct character classes I aim to predict.

A crucial aspect of my model's design is applying the SoftMax function in the output layer. SoftMax, a squashing role, transforms the model's raw output into a probability distribution across the various classes. This means that for each input data set, the model provides a set of probabilities, one for each probable character class, indicating the probability that the input image represents that character.

Thus, the SoftMax function allows my model to output a clear, interpretable prediction for each input: the character class corresponding to the highest probability is chosen as the model's prediction. This setup enables the model to translate complex graphic inputs into simple, understandable character predictions.

2.3. Method of generating the module

The application phase starts with utilizing Gaussian Blur, a technique applied to the graph for noise reduction [12]. The Canny Theory edge detection algorithm is then used to locate and highlight the graph's edges, which is the next step [12].

Subsequently, the graph is segmented into specific column-based slices. The dimensions of these slices and the total number of pieces in each row are calculated, considering the aspect ratio of the training data.

Once the graph has been sliced, these segments are transformed into an array and converted into a list. This list serves as the input for my trained Convolutional Neural Network (CNN) model for the prediction phase.

Upon the completion of the prediction, the output is translated back into corresponding characters based on each predicted class. These characters are assembled into an array that maintains the original layout and structure.

The final step involves converting this character array into a multiline text. This text is the ASCII representation of the original graph, signifying the successful application of the trained Convolutional Neural Network (CNN) model.

3. Experiment

3.1. Method of preparing the data

A collection of 5789 TrueType fonts (TTF) was amassed for this study. The fonts were sourced primarily from the Ubuntu operating system's installed fonts and various font repositories hosted on GitHub, such as Google Fonts, Nerd Fonts, and others.

I identified 95 classes for data generation, choosing to use characters within the decimal range of 32 to 126 according to their Unicode (UTF) codes. The frequency of data for each character ranged between 3700 and 5755 instances, with the underline character having the least data at 450 instances.

Each character from each font file was rendered with a spacing of 5 pixels from the edge on a 100x100 pixel graph. Edge detection was then performed on each character graph using the Canny method [12], and the resulting data was preserved in an array format for further processing in the study.

3.2. Training

For step 1, I built the structure of the employed deep learning model for this study is derived from the InceptionV3 model [13], a popular pre-trained Convolutional Neural Network (CNN) model initially devoid of any pre-loaded weights. The classifier's SoftMax activation function was added, while the top fully linked layer was removed. The pooling method, the number of classes, and the input shape were all tailored to my unique dataset.

For step 2, a layer with a global maximum pooling was added, and then a dense, fully linked layer with 1024 units and a ReLU (Rectified Linear Unit) activation function was added. The final layer was another dense layer comprising several units equivalent to my class count and implemented with a SoftMax activation function [4].

For step 3, the Adam optimizer was used when the model was compiled, and the loss function was defined as sparse categorical cross-entropy. Model performance was gauged using the accuracy metric [6].

For step 4, my data set was split in an 80/20 ratio into training and validation sets. A random seed was established to guarantee the model's repeatability.

For step 5, the validation accuracy was closely monitored during the training process, saving only the weights of the best-performing model.

For step 6, the model was trained using a batch size of 1024 over a period of 30 epochs. The model was tested after training on a validation set, resulting in the final loss and accuracy scores.

For step 7, the last step of this part, the model and its weights were saved for future usage.

3.3. Generation

The core process of my study involves utilizing the previously trained Convolutional Neural Network (CNN) model in InceptionV3 architecture to predict characters from graphical representations [12].

The initial step involves applying the Canny edge detection method on the input graph to highlight its edges [12]. This approach accentuates the boundaries within the graph, contributing to a more refined demarcation of potential character blocks for my model to analyze.

After edge detection, the graph is split into discrete columns. The size of each slice is determined based on the dimensions of my training dataset to align with my Convolutional Neural Network (CNN) model's input requirements [2]. This segmentation transforms the input graph into separate regions, each likely containing a character.

These slices are subsequently reshaped into arrays, conforming to a structured format that matches the input expectations of my Convolutional Neural Network (CNN). This restructured data is fed into my trained Convolutional Neural Network (CNN) model for character prediction.

The output from the prediction process is a series of probability distributions (SoftMax predictions) across potential characters. I convert these probability distributions into actual character predictions by selecting the character associated with the maximum probability value in each distribution.

Each predicted character is then mapped back to its originating block in the input graph. I add a space between each character to enhance readability, helping preserve the graph's overall coherence and interpretability.

The final output is a character-populated graph, which represents the result of my computational pipeline: a transformation of the input graph into an arranged set of predicted characters. This result is a testament to the predictive capabilities of my trained Convolutional Neural Network (CNN) model.

4. Result of the experiment

4.1. The result of training convolutional neural network (CNN)

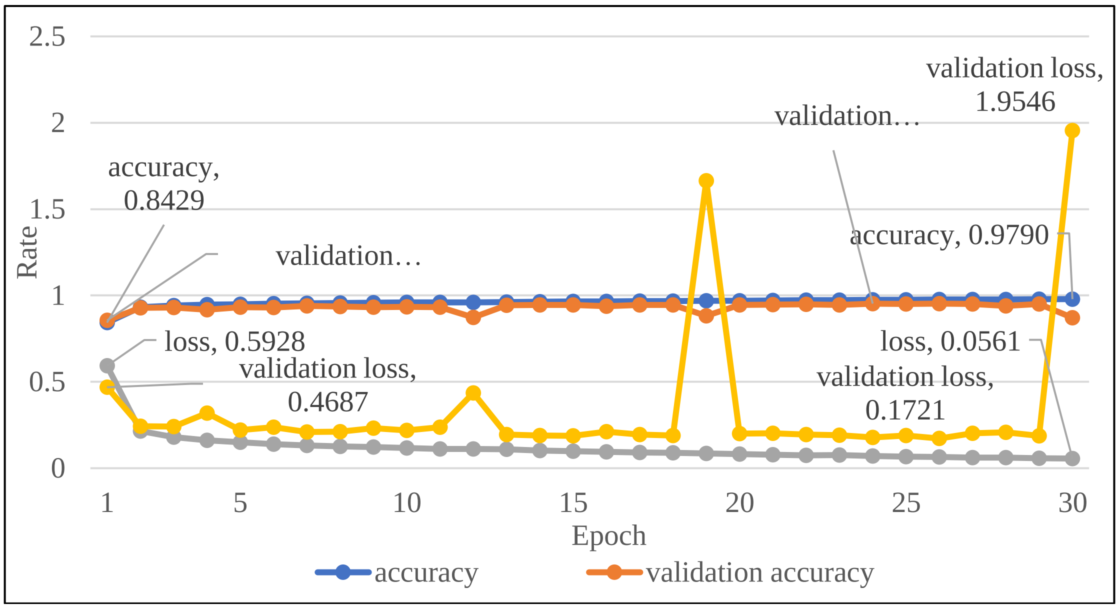

Figure 2. The history of training Convolutional Neural Network (CNN).

The proposed model was trained over 30 epochs. The training accuracy and loss and validation accuracy and loss were recorded for each epoch, represented in Figure 2 above.

The training accuracy showed an upward trend, starting from about 0.8429 in the first epoch and ending at about 0.9790 in the last epoch. This shows that the model's accuracy in accurately classifying the treated images in the training set increased over time. Meanwhile, the training loss showed a downward trend, starting from about 0.5927 in the first epoch and ending at about 0.0561 in the last epoch, confirming the model's improving proficiency.

In order to evaluate how well the model performed on unobserved data and to avoid overfitting, the validation accuracy and loss were tracked concurrently. The validation accuracy started at about 0.8556 and peaked at 0.9531 at the 24th epoch but experienced fluctuations afterward. This indicates that the model did not perform as consistently on the validation set as it did on the training set, which may be an indication of overfitting.

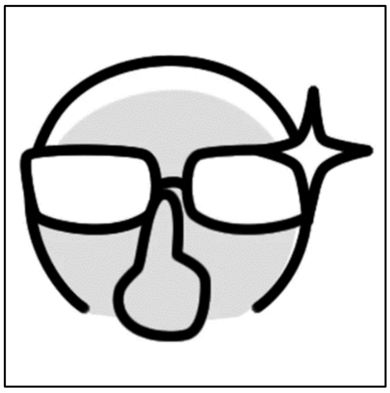

Figure 3. Original test graph.

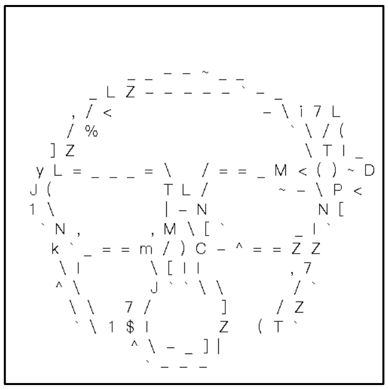

Figure 4. Translated graph with setting column number to 20.

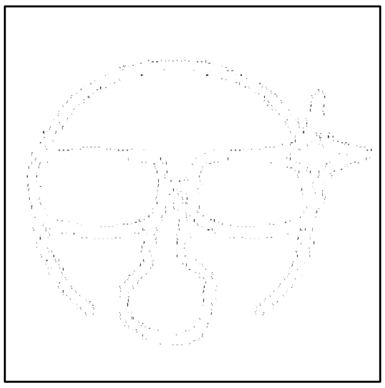

Figure 5. Translated graph with setting column number to 100.

The validation loss started at about 0.4687 and reached its lowest value at the 26th epoch (about 0.1721). After that, it fluctuated significantly and even showed an extreme value at the last epoch (approximately 1.9546), demonstrating an issue that needs further investigation and optimization.

Overall, the model demonstrated reasonable proficiency in the training data but showed signs of possible overfitting. Future work could include methods to mitigate over fittings, such as dropout, early stopping, or more extensive data augmentation.

4.2. The result of generation

My analysis delved into the nuances of character representation for graphical images (Figure 3.), considering the number of columns used for slicing the input graph. I conducted experiments using two distinct scenarios: one with a lower number of columns (Figure 4.) and the other with a higher number of columns (Figure 5.).

My system effectively represented simpler geometrical shapes, such as triangles using character symbols when operating with a lower column count. However, its proficiency diminished with more complex line structures, which may be lost with fewer columns due to the larger informational content these structures encapsulate.

Contrarily, using a higher number of columns allowed for a richer character representation of the input graph. Despite some instances where the chosen characters may not seem to be the optimal choice for a particular portion of the graph, it is worth noting that the Convolutional Neural Network (CNN) model has discerned some similarities between the character and the graph slice, such as comparable curvature patterns. This is a byproduct of the model's training process, where it learns to recognize and encode various graphical features into potential character counterparts.

These findings underscore the importance of column number optimization in the transformation process. By tuning the number of columns, I can influence the granularity of character representation and improve the graphical fidelity of my model's output. As such, the system can partially run as I expect. But striking the right balance between simplicity and precision is crucial for a nuanced graphical interpretation.

5. Conclusion

In this study, a distinctive approach is proposed for transforming graphical images into corresponding character layouts using a Convolutional Neural Network (CNN), a method backed by a wealth of research in deep learning. The findings suggest the model's potential to formulate a coherent character-based graph encapsulating essential elements of the graphical representation. However, the granularity of the slicing process significantly impacts the process's accuracy, as evident in the varying outputs produced with different column numbers. This research could expand into various domains to enhance the model's performance and applicability. For example, fine-tuning the slicing column count could create a balance between the simplicity and precision of the resulting character graph, thus boosting the model's effectiveness. An intriguing direction for future research would be to expand the model's training set to include a broader array of graphical patterns and associated character mappings. This approach would improve the model's versatility, enabling it to handle more complex and diverse graphical formats. Implementing advanced image processing methods, such as edge detection algorithms and pattern recognition, can enhance the preprocessing phase and result in more accurate character predictions. Despite the innovative and effective approach proposed in this study, there's ample room for advancement. This investigation into graphical-to-character translations marks a new era in data representation, with potential applications in data compression to the creation of novel graphical interfaces. The field of machine learning and image processing continues to evolve [1], which implies that there's significant potential in this research area. Consequently, advancements in these fields will substantially influence the capability to develop and refine the methodologies and findings presented in this study.

References

[1] Jordan, M. I., & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. Science, 349(6245), 255-260.
[2] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems, pp. 1097-1105.
[3] LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015 May;521(7553):436-444.
[4] He K, Zhang X, Ren S, Sun J. Deep Residual Learning for Image Recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition 2016. p. 770-778.
[5] Mayer-Schönberger, V., & Cukier, K. (2013). Big Data: A Revolution That Will Transform How We Live, Work, and Think. Houghton Mifflin Harcourt.
[6] Kingma DP, Ba J. (2014). Adam: A Method for Stochastic Optimization. arXiv preprint arXiv:1412.6980.
[7] Gatys, L. A., Ecker, A. S., & Bethge, M. (2015). A neural algorithm of artistic style. ArXiv, arXiv:1508.06576.
[8] Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2017). Grad-cam: Visual explanations from deep networks via gradient-based localization. Proceedings of the IEEE International Conference on Computer Vision, pp. 618-626.
[9] Johnson J, Alahi A, Li FF. Perceptual losses for real-time style transfer and super-resolution. In European conference on computer vision 2016 Oct 8 (pp. 694-711). Springer, Cham.
[10] Ulyanov D, Vedaldi A, Lempitsky V. Instance normalization: The missing ingredient for fast stylization. arXiv preprint arXiv:1607.08022. 2016 Jul 27.
[11] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. Neural networks, 61, 85-117.

[12] Canny J. A computational approach to edge detection. IEEE Transactions on Pattern Analysis and Machine Intelligence. 1986;PAMI-8(6):679-698.

[13] Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z. (2016). Rethinking the Inception Architecture for Computer Vision. arXiv preprint arXiv:1512.00567.