Residual Learning for Optical Character Recognition

Ryan Friberg
Columbia University, COMS 4995 Midterm Project
March 2nd, 2023

0 GITHUB REPOSITORY

https://https://github.com/ryan-friberg/COMS4995_OCR

1 INTRODUCTION & PROJECT OBJECTIVE

The primary objective of this project was to develop a computer vision pipeline that, given an intake of visual data or images, will be able to return any text data that may be present in that data. This task is more widely known as optical character recognition (OCR). The end result ended up being a combination of commercial and my own software including Google Tesseract for text identification and a deep residual neural network for character classification designed and trained from scratch. This paper outlines and documents the methods and procedures by which this pipeline was developed as well as the challenges faced along the way.

2 METHODS & MODELS

Pipeline Overview

The task of OCR can be broken down in two different sections. The first subtask is to identify where text in an image is, if it exists at all. The second is, given this region of the image, classifying what text is present. With this in mind, there are two feasible ways to approach OCR. One is to undertake the process of text detection and classification simultaneously (such an approach would necessitate the use of an advanced model such as a Contortionist Temporal Classification (CTC) model or some equivalent). The other approach is to break up OCR by subtasks and build models to individually do each task with a system of passing information between the two models.

Given the timeline of this project, the second approach was selected. To further simplify the task, this project utilizes a commercial method for text identification from Google called Tesseract. Using this package, the first stage of this OCR pipeline gives a sequence of single-character bounding boxes for every character that Tesseract recognizes within a given image. The second stage is a deep residual neural network built and trained from the ground up for the purpose of character identification. The pipeline extracts each of Tesseract's bounding boxes as a new image, applies the necessary image transformations, and then passes it to the residual network which gives a predicted character label for each bounding box. All together, the pipeline allows for OCR from a raw image that had no prior preprocessing.

Model Specifics

The residual network was designed based off of that which was described in *Residual Learning for Image Recognition* [1]. A residual network was specifically chosen so that the process of designing a new model would be relatively straightforward with a reasonable number of parameters (compared to something like a transformer) and that a deeper model could be achieved. A deeper model is desired as it would have a higher chance of being able to accurately encapsulate a classification task with so many labels in its latent space.

Given an image, it would return a predicted class of 62 possible options (0-9, a-z, and A-Z). In total, it consists of 107 layers with 12,953,566 trainable parameters. The architecture was designed as a sequence of residual blocks that individually function as a shallow convolution network with a residual tie back to its input. The residual ties allow for such a deep network to be constructed and trained without worrying about a vanishing gradient. The design diverged from the originally cited paper by having a different block architecture such that each block was both deeper and had more activation layers. The shapes of the input and output channels were selected based off of the given batch size (4), and the size of the images that the model would be given. After some minor experimentation, a total of seven residual blocks was chosen as it appeared to adequately encode the data over the high number of class labels (to be further discussed later in this paper).

The model was trained with various iterations of hyper parameters, but ultimately settled with a learning rate of 0.01, 2 data loader workers, batch size of 4, a 3/4 to 1/4 train-test split in the data, and 10 training epochs (with checkpointing). Along the way, the model was exclusively trained with cross entropy loss, but trained with different optimizers including standard SGD, Adadelta, and Adam to differing performance.

3 DATA

Overview

During development, in hopes of increasing the generalizability of the model, the choice was made to "increase the difficulty" of the task by training the residual network on completely different data than what would be passed through the pipeline. The residual network was trained on a conglomeration of *single character* data from the datasets within *The Chars74k*[2] which consisted of several sets of 128x128 images of individual characters. This dataset was picked due to its size, roughly 65k usable images, and its diversity, covering different hand writings, fonts, and numerous other differentiation's as well as the evenness in label distribution across the different classes (ie, there was no bias towards some characters' representations within the data over others). However, this data was not without its faults and a major one is that every image largely appear flat with a white background which was potentially a troublesome issue

for detecting data in real-life situations (such as appearing in different colors or in different textures, etc). Additionally, there seemed to be some organizational issues as there were many instances of incorrectedly-cased (upper or lower case) letters being present in the wrong class (ex. an image of J in the folder for j). While the best effort was made to manually move any that I happened to see, across tens of thousands of images this was bound to have a negative effect on training as this is directly confusing the model's classes.

As mentioned, the model was trained on a different set than it was "tested" on within the pipeline. The multi-character data that was passed to the model was derived from the *IIIT 5k* dataset [3] which consisted of formatted images of text with varying colors, resolutions, fonts, etc. Again, this data was cropped with the help of Google Tesseract and passed to the residual network.

Within the project, both of these have a custom dataset class (though one is a simple stripped down approach). Lastly, section 6 of this paper has an appendix with some examples of the training data used.

Labels

This data was broken down into 62 different categories, one for each possible character. For the single-character imageset, this is a trivial process but that is not the same for images with many characters. It is infeasible to map labels to words both due to the sheer volume of English words but also for the fact that non-words (including just random characters) and numbers may appear in the image and it would still be valid for the network to return those characters. To tackle this, one could reasonably add more advanced aspects to the classification task or preprocess the data and go through the Tesseract outputs and label them by hand. However, given the timeline of the project, a third option for labels was selected. When Tesseract gives the character bounding box information of a character, it conveniently also gives its own prediction for what that character is. Operating with the assumption that Google's commercial software will achieve a higher accuracy than my humble neural network, I simply extracted these values, withheld them from the residual network, and used them as the gold standard for each cropped image. This was ideal because each image was only given a label for all the text present, but there was no easy mapping between which characters would be in which bounding boxes. Additionally, the images included more characters than the 62 this project was aiming to identify (ex punctuation and -technically- spaces) which further obfuscated the label design process. While this approach is not a perfect solution, as Tesseract can also be wrong not only in terms of label but also the bounding box, it provides a simple and reasonably strong method for evaluation.

Transformations

A handful of transformations were applied to both datasets in order to improve the training and accuracy of the model. Examples include converting images to tensors and normalizing them to the mean and standard deviation of ImageNet to reduce outliers. Additionally, all data was resized to be 3x64x64 as 3x128x128 was needlessly large and the additional pixels largely did not provide an additional advantage in classification.

In order to attempt to add variation to the single-character data, transformations were added to randomly invert the images, adjust sharpness, and jitter the color in hopes of preventing the model accidentally associating characters with strictly black objects on white backgrounds. Similarly, the multi-character set had a grayscaling transformation applied to all the data because the color of the text also does not add any function in its classification.

4 RESULTS

This project was broken down into two different parts and as such, will have two different sets of results. One for character classification and one for the OCR pipeline's accuracy as compared to Google Tesseract. Additionally, there were two sets of training passes for this project, one with heavy data augmentation (those that were described in the prior section) and one with only minor transforms applied (which only consisted of normalizing to the ImageNet statistics).

Character Classification

Because the model has been trained over a set of data with even representation of every class among the 62 possible options, a simple accuracy measurement was chosen for the evaluation metric of the resnet. After training the model over different parameters, data augmentations, and with different optimizers, the highest accuracy achieved was roughly 86%. Interestingly enough, the optimizer did play a relatively significant role in the performance as training with the Adam optimizer never allowed the model to come even close to the observed peak obtained by other optimizers (hovering around a peak of 70% accuracy). The Adadelta optimizer ended up yielding the overall peak accuracy.

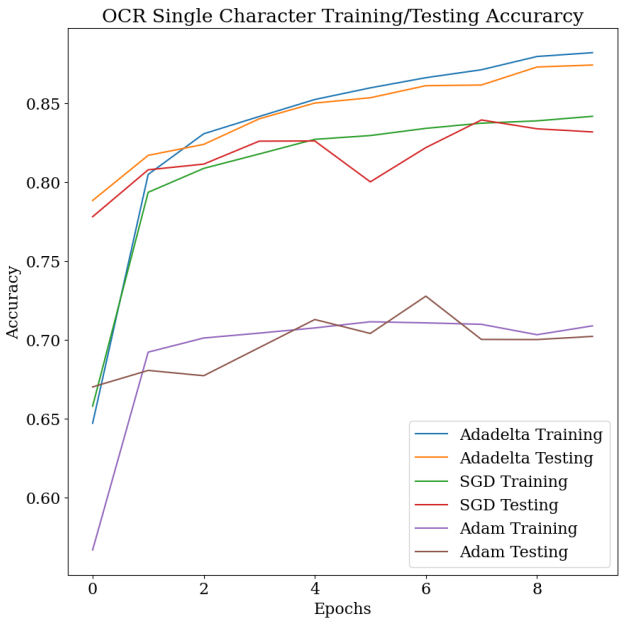


Figure 1. A graph of the accuracy of the model across three optimizers

Figure 1 shows the progress of the model on the highly augmented data throughout it's training epochs. Here, it is clear to see how the Adam optimizer lagged behind. Additionally, the training accuracy essentially monotonically increases with more epochs but after a certain point the test accuracy because somewhat volatile suggesting that over-fitting to the training data has begun.

OCR Results

The OCR section of this pipeline did not have any training. It was simply the application of two models put together. As such, the metric used was a simple, single forward pass of the data through the pipeline and accuracy comparison to Google Tesseract's predicted character labels. Using the resnet trained on the minorly augmented data resulted in a 30% accuracy by this metric and using the model trained on highly augmented data surprising yielded an increase in accuracy to about 39%. It is important to note that this is not a ground truth metric as Tesseract does not have a 100% accuracy.

5 DISCUSSION & FUTURE STUDY

It is clear that there was a ceiling on the training possibility of this task as the training runs of the model all seemed to plateau at some accuracy mark and have issues improving beyond that. Going into this project, I was unaware of what accuracy would be achievable both for the individual subtask of character recognition but also OCR as a whole. At first, the $\approx 39\%$ accuracy for the OCR pipeline compared to Google Tesseract was a bit of a disappointment but in the context of the problem, I do think it makes sense. 39% initially seems rather low, but randomly guessing would only yield a correct label 1.6% of the time which proves that some generalization between the datasets has occurred. In my opinion, the two main issues that enforced this ceiling were the labels and the data used.

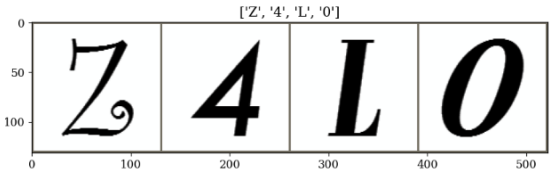
This problem was always going to have difficulty achieving high accuracy due to the sheer number of labels. 62 labels is quite a lot, and if there is some ambiguity in the model's prediction, there is a much larger chance of its guessing incorrectly, even if it has confidence in the classification down to a handful of options. Additionally, there are many labels that look too visually similar (ex. 0 vs O vs o or W vs w) which would be easy for a human to mix up. This problem was only confounded by the data misorganization as mentioned before. In general, getting high confidence over every prediction in this task is difficult. To combat this, an area of future study that would be interesting would be a weighted loss function such that classifying O as 0 will not get penalizes as heavily as S vs M for example.

It is my suspicion that the problem was, in fact, the data that was used. For this project, I was unable to find a solid-enough single character dataset that was composed of characters in the real world or in other contexts but in the future, I think taking more time to conglomerate and/or crop data from multiple datasets would have made substantial improvements. This is clear after the added image augmentation yielded almost a 10% increase in the pipeline accuracy. I abstained from training the resnet on the multi-character dataset both because there were far fewer images (5k vs 65k) with a less even label distribution, and I wanted to avoid having the model overfit to the testing data but instead become a generalized character classifier. In hindsight, this may not have been a bad idea in theory but I think it would be important to introduce more variations into the training data that extend beyond what the pytorch transformations were able to do. If the desired task consisted of focusing on more neatly documented images, such as screenshots of pdf text for example, I do believe that this pipeline would have a far higher OCR accuracy meaning it was the diversity in the image text data that likely held the model back. A model is only as powerful as the data it is trained on and there are always ways to improve.

Reflection

This project as a whole has been a big experimenting session. Many of these systems were foreign to me prior to beginning and working with new data and model architectures has given me a better picture of how everything works (and a much stronger appreciation for the importance of building datasets!). While not every decision I made had a strong theoretical reasoning backing it up, the fast-paced trial and error has helped me learn such that if I were to do it again, I believe I could build upon this experience and do it better.

6 APPENDIX



An example of a training batch passed to the resnet

An example of a multi character image passed to the OCR pipeline

An example of a multi character image passed to the OCR pipeline

7 REFERENCES

[1] https://arxiv.org/pdf/1512.03385.pdf

[2] http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/

[3] https://cvit.iiit.ac.in/research/projects/cvit-projects/the-iiit-5k-word-dataset

[4] https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/33418.pdf

[5] https://towardsdatascience.com/residual-network-implementing-resnet-a7da63c7b278

[6] https://www.run.ai/guides/deep-learning-for-computer-vision/pytorch-resnet

[7] https://pytorch.org/docs/stable/index.html

[8] https://github.com/tesseract-ocr/tesseract