Anthony Le & Nathan Traxler

CS 229: Machine Learning

Andrew Ng, Moses Charikar, Christopher Ré

13 December 2019

# Kuzushiji Character Recognition & Classifying

**Abstract:**

We were interested in developing an object detection and classifying model sophisticated enough to detect and translate ancient Kuzushiji characters in a given image. After preliminary research, we decided on using a Tensorflow Deep Neural Net for end-to-end text recognition and classification. We trained two different pre-trained neural net models. The model would be fed images and data files containing the labels and bounding boxes of a certain character on a page. After the model was fully trained, we tested the model on a validation set. We assessed the recognizer and classifier accuracy through their respective $F_1$ scores.

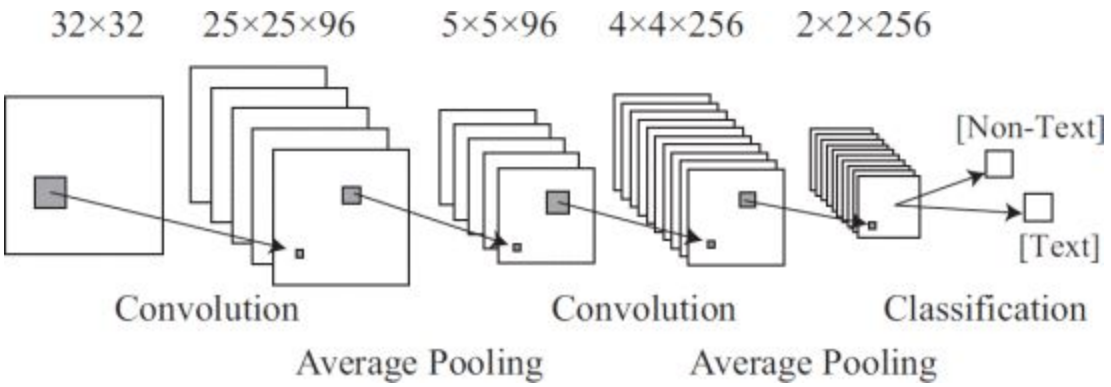
**Introduction:**

Kuzushiji is an ancient Japanese style of cursive handwriting that has fallen out of use in modern society. The surviving texts written in kuzushiji can only be read by about .01% of Japanese natives. We plan to take thousands of these texts, locate kuzushiji characters on the page, and correctly identify the corresponding modern japanese character. We are interested in this project as it could unlock thousands of stories, records, and texts that were previously hidden from a vast majority of society. This is a clear example of an application project of computer vision as the computer needs to be able to locate and recognize characters from an image. However, this could be extended to also apply to natural language processing as characters need to be identified and/or corrected in order to follow the rules of the Japanese grammar.

**Related Work:**

Studies done by professor Anderw Ng have shown that end-to-end text recognition is viable through large, multilayer neural networks and recent developments in unsupervised learning [1]. According to the study, CNN's have had similar success in tasks such as

handwriting recognition, character recognition, and visual object recognition. One can see that a CNN was a good choice of model for our object detection classifier.

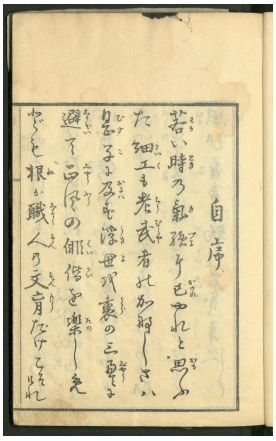



Another article has shown that incredibly similar research has been done, where an invariant recognition system built atop a fuzzy min-max neural network achieved 93.9% prediction accuracy when looking at chinese characters [3]. The neural net outperformed the K nearest neighbors and minimum mean-distance classifier by almost 20%, in this study, confirming our intention to use a deep neural net to build our model's object recognizer and classifier. The study had conceded that character classification of this complexity and variance would lead to a difficult challenge. The picture to the left shows the types of images this study trained on. They normalized each of the images containing only a single character because they were led to believe that normalization of the characters improved test accuracy. This led us to normalize our images.

**Dataset and Features:**

We used the Kuzushiji Recognition competition on Kaggle for our training and testing dataset. This dataset contained 3881 labeled images of Kuzushiji scripture. These labels gave the label of the selected Kuzushiji character's translation as well as the coordinates and dimensions of the bounding box containing the single character. This allowed us to generate tf records that were usable by TensorFlow. If we look at how many individual characters our model was trained on, we had 542,281 individual character, with 4787 unique characters being identified. The combined raw data totals was about 3.28 GB. Some of the pictures did not have any

Kuzushiji characters on them, so we deleted these training examples. The original pictures were had resolutions about the suggested 720 x 1280 pixels, so we scaled down the resolution of our training images and normalized them in order to make the model train faster. Here is an example of our training data before any preprocessing:



| filename | width | height | class | xmin | ymin | xmax | ymax |
|---|---|---|---|---|---|---|---|
| 100241706_00004_2 | 2404 | 3874 | U+306F | 1231 | 3465 | 1364 | 3518 |
| 100241706_00004_2 | 2404 | 3874 | U+304C | 275 | 1652 | 359 | 1721 |
| 100241706_00004_2 | 2404 | 3874 | U+3044 | 1495 | 1218 | 1638 | 1287 |
| 100241706_00004_2 | 2404 | 3874 | U+3051 | 220 | 3331 | 273 | 3422 |
| 100241706_00004_2 | 2404 | 3874 | U+306B | 911 | 1452 | 972 | 1544 |
| 100241706_00004_2 | 2404 | 3874 | U+306B | 927 | 3445 | 998 | 3537 |

**Methods:**

We trained 2 CNN models obtained from the Tensorflow model zoo github on our processed text images. The first being the SSD MobileNet model because it is the smallest model with respect to number of parameters that needed to be trained as well as the least complexity. We began with this model in order to get a working implementation done quickly considering this model would train quicker than the others in the model zoo. The SSD MobileNet is meant for machines with low processing power such as Mobile devices. It works to limit the computation necessary to function as a CNN. It does this by running a depthwise convolutional layer on the input images running convolution on individual channels of the image instead of the RGB channels as a whole. It then runs through a 1X1 convolutional layer to save correct the dimensions of the output. This leads to a total computational reduction of $\frac{1}{D}$ where D is the kernel size.

We then moved to training the faster R-CNN inception model. The faster R-CNN model has shown some of the best results in object detection in recent years. This model builds upon two models before it, namely, the R-CNN and the fast R-CNN. The main improvement seen in faster R-CNN is the region proposal network (RPN) which outperforms the search selective algorithm seen in the previous models. The RPN proposes regions that may have objects in them using classification (whether the region contains any object) and box regression. The results in this step are fed to a CNN for classification and box regression.

We needed to use the softmax function in our output layers for the classification as we had to classify between 4787 different Japanese characters. The softmax function squashes the output so that it sums to one in order to give us probabilities. Since the softmax function is used to help us predict which character is in our bounding box it highlights the difficulty of this task.

**Experiments/Results:**

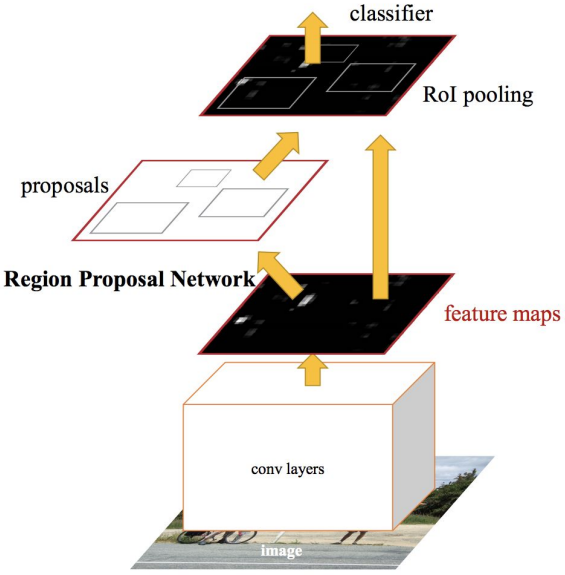
Considering this is a classification problem at heart, we looked at our $F_1$ score to measure performance. The $F_1$ score is the harmonic mean of precision and recall.

| Model | SSD MobileNet | Faster R-CNN Inception V2 |
|---|---|---|
| Training Precision | .59 | .72 |
| Training Recall | .57 | .70 |
| Training $F_1$ Score | .58 | .71 |
| Test Precision | .56 | .71 |
| Test Recall | .54 | .68 |
| Test $F_1$ Score | .55 | .69 |

Clearly, the Faster R-CNN performed much better than the SSD MobileNet which is to be expected as it is a much more robust model. These results are significant and likely useful in the translation process from kuzushiji to modern Japanese since there are currently so few people that are able to do this translation, however it is unlikely that this model would do an effective job relative to a human translator as it is expected they could do a translation of characters nearly perfectly. Lastly I would say our data is not overfit as the numbers for the training set and the test set are similar.

**Conclusion/Future Work:**

In summary, the faster RCNN inception v2 COCO model performed better than the SSD MobileNet. We believe that this model performed better during testing because it is not a single shot detection model and looks at the image multiple times when first detecting and then classifying the image. SSD Mobilenet combines the prediction of bounding boxes and classification probabilities into one convolutional network . Faster RCNN divides the tasks by first predicting the bounding boxes using a neural network, and then second predicting the class probability using a separate network. This division of processes tends to work better for smaller objects within a given image, so this can explain why the faster RCNN inception model outperformed SSD MobileNet. In terms of future work, we would like to create an end-to-end Kuzushiji character recognizer and classifier that works with a variety of Kuzushiji documents and can correctly translate an entire page with a test accuracy of 95%.




**Contributions:**

Our project was a combined effort of research, coding, and experimenting. Both of us worked to find a dataset for a problem we would like to tackle. We started this project knowing that we wanted to do something with computer vision and when we found the Kuzushiji data set we were highly content. Both of us implemented the model in attempts both locally and on Google Cloud. The work was divided from here as follows:

**Anthony:** Researched neural network architecture in order to find the best open source libraries available to create an object recognizer and classifier. Looked into how to verify our results using cross validation, confusion matrices, and $F_1$ scores.

**Nathan:** Worked on data manipulation and image pre-processing to aid our CNN, developing python scripts to aid in the preliminary stages of our model. Developed a preliminary classifier based on Keras models of Tensorflow. Implemented overall architecture of the Tensorflow CNN pre-trained models.

**Link to Code:**

https://drive.google.com/drive/folders/1OyPHGcimdQ1eAMSnQpBFFiVY0O88RA-X?usp=sharing

**References:**

[1] T. Wang, D. J. Wu, A. Coates, A. Y. Ng, End-to-end text recognition with convolutional neural networks. *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012).*, 3304-3308 (2012).

[2] Tensorflow. "Tensorflow Detection Model Zoo." GitHub, Google, 2019

[3] Chiu, Hung-Pin, and Din-Chang Tseng. "Invariant Handwritten Chinese Character Recognition Using Fuzzy Min-Max Neural Networks." *Pattern Recognition Letters*, North-Holland, 19 May 1998, www.sciencedirect.com/science/article/pii/S0167865597000299?via%3Dihub.

[4] EdjeElectronics."EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train -Multiple-Objects-Windows-10." *GitHub*, 13 Oct. 2019,github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10.

[5] Gandhi, Rohith. "R-CNN, Fast R-CNN, Faster R-CNN, YOLO - Object Detection Algorithms." *Medium*, Towards Data Science, 9 July 2018, towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e.