Object Detection with Neural Network and Faster-RCNN

Ziqi Zhao
Ningbo Binhai International Cooperative School
Email: zqzhao0626@outlook.com

***Abstract* – Neural Network is a system of neurons used for solving problems. Usually, it can be used in hand-writing recognition. Faster-RCNN is one of the most popular deep learning models for object detection. It was developed several years ago by Shaoqing Ren et al. It can train and test datasets. It functions well generally, especially in position detection, but sometimes has limitations such as its accuracy of classification still needs improvements. This paper will focus on the utilization of Faster-RCNN model to realize the detection, localization, and prediction of Kuzushiji characters.**

I. Introduction

Kuzushiji, a cursive writing style that was used to compile millions of invaluable books and records, is now facing a serious problem that for more than 99.9% of modern Japanese natives, understand Kuzushiji look like a daunting task.

Faster-RCNN is one of popular deep learning models that have remarkable performance in application of object detection; it is suitable for recognizing a Kuzushiji character. A basic form of neuron network includes an input layer, several hidden layers, and an output layer. By using this structure, the neuron network could effectively turn an area with fixed pixels in an original image from neurons that contain activations to represent grayscale values to neurons that contain activations to represent the possibility that a certain Kuzushiji character corresponds with the region.

To make this process more specific, what happens in the hidden layers is separating the certain region that needs to be recognized into smaller pieces with identified sides of part of the Kuzushiji character and using formula as shown below to manipulate how the output of a certain layer may become the input of the next layer and to what degree the layer may influence the next layer:

$$y = w * x + b$$

where y represents the output activation in a neuron, w represents the weight, and b represents the bias.

In this concept, weight, the number people assign to each neutron, is used to calculate how important a specific shape is in order to compose a crucial part of a specific Kuzushiji character in the next layer. Weights update is based on Backpropagation algorithm. Backpropagation, short for "backward propagation of errors," is an algorithm for supervised learning of artificial neural networks using gradient descent. Given an artificial neural network and an error function also known as loss function, the method calculates the gradient of the error function with respect to the neural network's weights. The algorithm will try to find the suitable weights which make the error function small as possible as it can.

And bias, the number people used to add after getting the sum of all the products of activations and weights of every connection between a particular neuron in a former layer and a fixed neuron in the next layer, is used to limit to what degree can cause the fixed neuron to be meaningfully activated. Then comes the softmax activation functions as shown below:

$$\sigma(\mathrm{z}) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{x^T w_k}}, \text{ for } j = 1 \ldots K.$$

The code below could also explain how it works:

```
logits = [2.0, 1.0, 0.1]
import numpy as np
exps = [np.exp(i) for i in logits]
```

By using softmax activation functions, we use a gradient-based optimization method to avoid the existence of negative log.

One of the most commonly-used softmax activation functions is sigmoid function as shown below; it guarantees that the activation function properly represents a possibility between zero and one:

$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

The sigmoid function is always used as a factor in the activation function—it is put into use after finishing all the calculation processes related to weights and biases. Since the outputs of layers can be any real number, it is particularly crucial to use the sigmoid function to turn the outputs into numbers between 0 and 1, which are well-prepared to be the inputs of the next layer.

Consequently, by using this activation function in neuron networking from time to time, people could successfully determine the input of every single neuron in the next layer from the former layer.

Because properly twist a multitude of weights and bias makes Kuzushiji-recognizing an extremely challenging task as the number of neurons in each layer increases, we could also use the matrix as shown below to substitute the activation function:

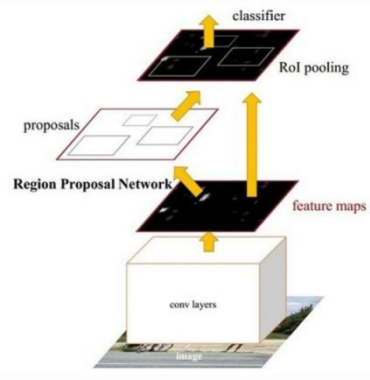
$$\begin{bmatrix} w_{0.0} & \cdots & w_{0.n} \\ \vdots & \ddots & \vdots \\ w_{k.0} & \cdots & w_{k.n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} = \begin{bmatrix} ? \\ \vdots \\ ? \end{bmatrix}$$

where all activations in one layer is a vector and all weights between the connections of this layer with a particular neuron in the next layer. Sigmoid works well for a classifier, because approximating a classifier function as combinations of sigmoid is easier than other functions, and it will lead to faster training process and convergence.

Thus, we successfully make the training and testing process in NN Model possible.

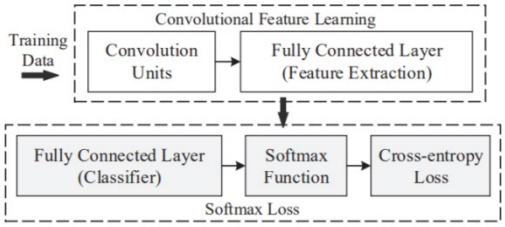
II. Materials and Methodology

The main process we use for training and testing data is through Faster-RCNN. Faster-RCNN is an object detection architecture. We can mainly separate it into three modules: Convolution Neural Network, Region Proposal Network (RPN), and Classes and Bounding Boxes Prediction. The whole architecture picture is shown as below:



(Figure 1)

And the process structure including calculation processes is shown below:



(Figure 2)

i. Convolution Neural Network

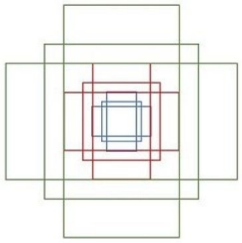
The first process in Faster-RCNN, convolution, is basic object detection and feature extraction. It is composed of three layers: convolution layers, pooling layers, and fully connected layers. In order to realize edge detection, a filter with a fixed number in each part of it is used to cover the original image with fixed pixels. Then, we calculate the products of the pixel values and the values shown on the filter with corresponding positions and sum them up. After that, we move the filter one-step right if we assign stride as one (if "stride=2", move two-steps right, etc.) and keep calculating sums until the filter covered all aspects of the picture. At this point, we could put all the sums together and get a new matrix, and thus successfully recognize some certain characteristics in the picture. Then come the pooling layers. The main purpose of it is avoiding over-fitting. Since the result we get from convolution layers are still containing plenty of parameters, it is inconvenient for people to get the main, crucial characteristics of the region in the picture. Two main pooling ways are max pooling and average pooling. The biggest difference between convolution and pooling, regardless whether it is max pooling or average pooling, is that although we still use a fixed size filter to get a new matrix and thus reducing the size of the matrix, the filter we used in pooling layers is blank, which means instead of multiplying and summing, we directly get the biggest number in the filter as a number of the new matrix—when it is max pooling—or get the average number in the filter—when it is average pooling. What follows the pooling layers is a fully connected layer. In this layer, each channel does not contain a matrix anymore, but instead, a number. Consequently, we could successfully detect and classify the object.

ii. Region Proposal Network

In Faster-RCNN, what follows the last convolution layer is RPN. RPN has a classifier and a regressor. The classifier helps people to determine the probability a proposal has a target object and the regressor helps people to get the proper coordinates of the object on the original image. RPN slides on the feature map, sets anchors, builds regressions for object classification and position detection. Firstly, we can judge whether an area contains an object or not by using loU value as shown below:

$$\text{loU} = \frac{\text{An Gt}}{\text{Au Gt}} \begin{cases} > 0.7 = object \\ < 0.3 = not\ object \end{cases}$$

where A represents anchors, Gt represents ground-truth boxes from the dataset, and u represents the union. Thus, by assigning nine anchors (shown in figure 1), a group of matrixes generated by "rpn/generate_anchors.py", as the basic detection windows to every single point on feature maps, RPN could eventually judge which anchors are positive anchors that contain the object, and which anchors are negative anchors that do not contain the object.



(Figure 3)

iii. Classes and Bounding Boxes Prediction

Similar to those in RPN, classes and bounding boxes prediction help classification loss and bounding-box regression loss. Nevertheless, there are no need to detect the object and the position of it, but rather, identifying that whether the region proposals propose the exact objects we are looking for and determining which class the object belongs to and refine the position suggested in the proposals.

III. Evaluation and Discussion

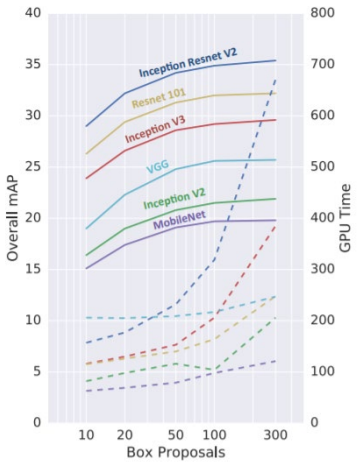
We used Faster-RCNN at first, but then found out that its modeling performance is not ideal since the softmax-based model can't caught the true probability distribution for so many classifications and has a bottleneck. The vectors used in the softmax and the output layers sometimes were too small to be the representative of true parameters of vocabulary sizes. In addition, the accuracy of Faster-RCNN is not ideal when we use it to classify objects and the running time is particularly long as well. So we tried some feature extractors as shown below to increase the classification accuracy:

Model	Top-1 accuracy	Num. Params.
VGG-16	71.0	14,714,688
MobileNet	71.1	3,191,072
Inception V2	73.9	10,173,112
ResNet-101	76.4	42,605,504
Inception V3	78.0	21,802,784
Inception Resnet V2	80.4	54,336,736

(Figure 4)[1]

By trying all different kinds of models listed above, we found out at the end that Resnet exactly is the right model for classifying Kuzushiji characters because it provides the most accurate result.

[1] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, Kevin Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. Google Research, April 2017. https://arxiv.org/pdf/1611.10012.pdf



(Figure 5)[2]

The basic structure of ResNet can be defined:

```
def __init__(self, block, layers, num_classes=1000,
    zero_init_residual=False):

    super(ResNet, self).__init__()

    self.inplanes = 64

    self.conv1 = nn.Conv2d(3, 64, kernel_size=7,
    stride=2, padding=3, bias=False)

    self.bn1 = nn.BatchNorm2d(64)

    self.relu = nn.ReLU(inplace=True)

    self.maxpool = nn.MaxPool2d(kernel_size=3,
    stride=2, padding=1)

    self.layer1 = self._make_layer(block, 64,
    layers[0])

    self.layer2 = self._make_layer(block, 128,
    layers[1], stride=2)

    self.layer3 = self._make_layer(block, 256,
    layers[2], stride=2)

    self.layer4 = self._make_layer(block, 512,
    layers[3], stride=2)

    self.avgpool = nn.AdaptiveAvgPool2d((1, 1))

    self.fc = nn.Linear(512 * block.expansion,
    num_classes)


  def forward(self, x):

    x = self.conv1(x)

    x = self.bn1(x)

    x = self.relu(x)

    x = self.maxpool(x)


    x = self.layer1(x)

    x = self.layer2(x)

    x = self.layer3(x)

    x = self.layer4(x)

    x = self.avgpool(x)

    x = x.view(x.size(0), -1)
```

[2] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, Kevin Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. Google Research, April 2017. https://arxiv.org/pdf/1611.10012.pdf

We config some parameters of the model in advance. For example, for Faster RCNN, we resize the input images in 512*512 and keep aspect ratio resizer: min dimension: 574 and max dimension: 764. For the model of feature extractor, anchor generator, the initializer and train processing, we also have special config in advance, details can be check in Github, file: faster_rcnn_resnet101.config.

For this project, we have known how deep learning can solve a practical problem in real life and how to finish a task as a team. Since then, I have a further understanding of Python and Neural Networks. I knew how to use Faster-RCNN and Convolutional Neural Networks to achieve object detection, and have the rudimentary ability to create small programs to successfully recognize Kuzushiji characters. Besides, by working as a team to finish the task, I gradually learned how to cooperate with other people to conquer difficulties one by one and get along with people with different points of view. By joining the Kaggle Big Data Contest, I found my potential in the field of data and decided to choose data science as my college major. And this project truly prepared me to be a qualified future financial analyst.

IV. References

[1] Achraf Khazri. Faster RCNN Object detection. April 2019. https://towardsdatascience.com/faster-rcnn-object-detection-f865e5ed7fc4 (Accessed on Thursday, Oct 24, 2019; 11:37)

[2] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. Microsoft Research. https://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf (Accessed on Thursday, Oct 24, 2019; 11:42)

[3] Tomasz Grel. Region of Interest Pooling Explained. Feb 2017. https://deepsense.ai/region-of-interest-pooling-explained/ (Accessed on Thursday, Oct 24, 2019; 23:22)

[4] Sekitoshi Kanai, Yasuhiro Fujiwara, Yuki Yamanaka, Shuichi Adachi. Sigsoftmax: Reanalysis of the Softmax Bottleneck. NeurIPS, 2018. https://papers.nips.cc/paper/7312-sigsoftmax-reanalysis-of-the-softmax-bottleneck.pdf (Accessed on Thursday, Oct 24, 2019; 23:25)

[5] Weiyang Liu, Yandong Wen, Zhiding Yu, Meng Yang. Large-Margin Softmax Loss for Convolutional Neural Networks. http://proceedings.mlr.press/v48/liud16.pdf (Accessed on Thursday, Oct 24, 2019; 23:35)