Deep Learning CAN Report

Hanwen Liu 522030910109

June 3, 2024

1 Introduction

Handwritten mathematical expression recognition (HMER) is an important task of document analysis, which involves a wide range of applications such as assignment grading, digital library service, and office automation. The task is to convert an image of handwritten mathematical expression into its corresponding markup language (such as LaTeX). Despite the appearance of modern method such as OCR, HMER is still to be improved due to its complex formulas and various writing styles.
The main architecture used in HMER nowadays is encoder-decoder architectures. The architecture formulates the HMER problem into an image-to-sequence translation problem. The encoder first projects the input image of strokes in online handwriting $\mathbf{x}$ to a high dimension embedding $\mathbf{e}$. Then, the decoder generates the output hypothesis $\hat{y}$ with attention mechanism combined. However, insufficiency still exists: the accuracy of attention is still not guaranteed when meeting complex formula.
In the paper, author proposed an unconventional Counting-Aware Network CAN, which combines two complementary tasks: HMER and counting. With adopting DWAP and ABM as the baseline network, CAN achieves SOTA recognition accuracy on CROHME dataset. Therefore, the method author proposed can be generalized to various encoder-decoder models, bringing them better robustness and accuracy.

2 CAN Model

2.1 Overview

Counting-Aware Network(CAN) consists of three major components: a backbone network, a multi-scale counting Module(MSCM) and a counting-combined attentional decoder(CCAD) as shown in Figure 1.

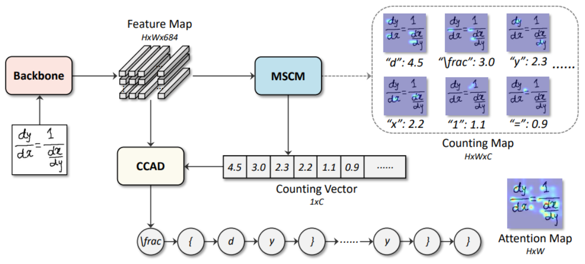

Figure 1: Structure of CAN

The backbone network adopted in the paper is DenseNet, an extension to the traditional Convolution Neural Network, in which every layer is strongly connected. The DenseNet first extracts the feature map $\mathcal{F} \in R^{H \times W \times 684}$ of the input image $\mathcal{X} \in R^{H' \times W' \times 1}$, where $\frac{H'}{H} = \frac{W'}{W} = \frac{1}{16}$. Then the feature map was used by MSCM to predict the number of each symbol class and generate the counting vector $\mathcal{V}$ which represents the number of each symbol. The feature map and the counting vector are then fed into CCAD to generate the predicted output $\hat{y}$.

2.2 Multi-Scale Counting Module

The multi-scale counting module(MSCM) is proposed to obtain the counting vector V, which is further used in the decoder to predict the output $\hat{y}$. The structure of MSCM is shown in Figure 2. MSCM adopts a multi-scale feature extraction method due to various symbol size. Specifically, MSCM uses two kernels of different sizes(3×3 and 5×5) to extract multi-scale features parallelly. Following this method, neither large symbol nor small

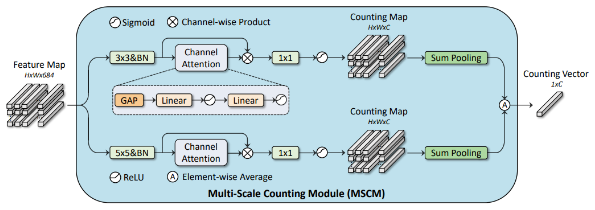

Figure 2: Structure of MSCM

symbol would be easily missed. The outputs of two convolution layers are then enhanced by the channel attention:

$$\mathcal{Q} = \sigma(W_1(\mathcal{G}(\mathcal{H}) + b_1),$$
$$\mathcal{S} = \mathcal{Q} \otimes g(W_2\mathcal{Q} + b_2),$$

where $\mathcal{H} \in R^{H\times W\times C'}$ is the extracted feature map from the convolution layer and $\mathcal{S}$ is the enhanced feature from the channel attention.
An 1×1 kernel convolution and a sigmoid function are then utilized to convert the enhanced feature map $\mathcal{S} \in R^{H\times W\times C'}$ to the counting map $\mathcal{M} \in R^{H\times W\times C}$, which is a combination of C pseudo density map $\mathcal{M}_i \in R^{H\times W}$ reflecting the position of i-th symbol class. According to the method used in counting number by density map, author feeds the two counting maps into sum-pooling layer to get two counting vectors $\mathcal{V}^{3\times3}$, $\mathcal{V}^{5\times5} \in R^{1\times C}$:

$$\mathcal{V}_i = \sum_{p=1}^{H}\sum_{q=1}^{W} M_{i,pq}$$

where $\mathcal{V}_i \in R^{1\times1}$ represents the predicted number of i-th symbol. Finally, the final counting vector $\mathcal{V}^f \in R^{1\times C}$ is obtained by:

$$\mathcal{V}^f = \frac{\mathcal{V}^{3\times3} + \mathcal{V}^{5\times5}}{2}$$

The output counting vector $\mathcal{V}^f$ is the final output of MSCM and is later fed into CCAD to generate the predicted formula.

2.3 Counting-Combined Attention Decoder

In order to adjust the traditional decoder to the counting vector $\mathcal{V}$, author proposed a new decoder architecture called counting-combined attention decoder(CCAD), which is shown in Figure 3.

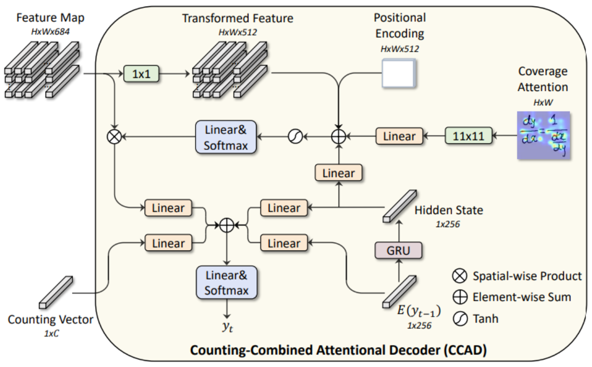

Figure 3: Structure of CCAD

The input feature map $\mathcal{F} \in R^{H\times W\times 684}$ is first transformed to $\mathcal{T} \in R^{H\times W\times 512}$ by a 1×1 convolution kernel. Following the convolution, the transformed feature map $\mathcal{T}$ is then combined with a spatial positional encoding $\mathcal{P} \in R^{H\times W\times 512}$ to enhance model's awareness of spatial position. The new combined feature map is finally put together with the hidden state $h_t \in R^{1\times256}$ and the sum of all past attention weights $\mathcal{A}$ to obtain the attention weights $\alpha_t \in R^{H\times W}$:

$$e_t = w^T tanh(\mathcal{T} + \mathcal{P} + W_a\mathcal{A} + W_h h_t) + b,$$
$$\alpha_{t,ij} = exp(\alpha_{t,ij}) / \sum_{p=1}^{H}\sum_{q=1}^{W} e_{t,pq},$$

where the hidden state h_t is obtained by passing the embedding symbol $y_{t-1} \in R^{H\times W\times 256}$ into a GRU cell and coverage attention $\mathcal{A}$ is the sum of all past attention weights.
The obtained attention weights α_t and the feature map $\mathcal{F}$ are then spatial-wise multiplied to get context vector $\mathcal{C} \in R^{1\times256}$, which in last used to calculate the final predicted output y_t with counting vector $\mathcal{V}$, embedding symbol y_{t-1} and hidden state h_t :

$$p(y_t) = softmax(w_o^T(W_c\mathcal{C} + W_v\mathcal{V} + W_t h_t + W_e E) + b_o),$$
$$y_t \sim p(y_t),$$

Apart from traditional decoder, CCAD introduces counting vector $\mathcal{V}$ to better combine global information, which is not equipped by context weight, embedding symbol and hidden state. The introduce of counting vector boost the accuracy and robustness of the encoder-decoder architecture and is the key of the CAN model.

3 Datasets

The datasets used in the paper is CROHME dataset, which is a widely-used dataset in HMER with 111 number of classes of symbol. The training set contains 8836 handwritten mathematical expressions. The testing set includes three CROHME dataset: CROHME 2014, 2016, 2019 with over 3000 handwritten mathematical expressions.

4 Algorithm Replication

4.1 Preparatory Work

According to the paper, I download the source code from author's official github page via git clone command:

git clone https://github.com/LBH1024/CAN.git

Following the git clone command is downloading numerous required python libraries, including pytorch, thop, tensorboard...This can be done with pip install command such as pip install torch. The dataset of CROHME can be downloaded in author's BaiduNetdisk.
Additionally, a powerful GPU with at least 32GB RAM is the basic hardware requirement. Author uses a single Nvidia Tesla V100 GPU to train the model. Similarly, in my replication of the algorithm, I mainly use a single Nvidia 4080s GPU to train and test the CAN model.
To start to train the CAN model with CROHME dataset, one more step is required, the path of dataset CROHME and the path of config.yaml should be modified to adjust local environment.

4.2 Training and Testing

After all the preparetory work is done, run train.py in the folder and the process of training and testing is shown clearly in the screen. I trained the CAN model with batch size 8 and the Adadelta optimizer for up to 175 epochs and after training in each epoch, testing will be performed to test the WordRate and ExpRate of the trained model.

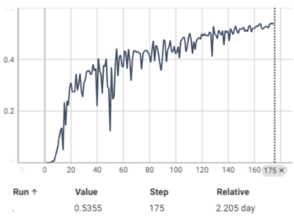

Figure 4: ExpRate

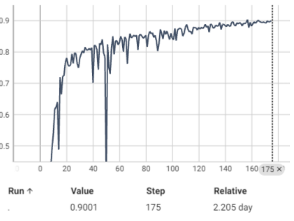

Figure 5: WordRate

WordRate represents the percentage of correctly recognized words in handwritten mathematical expressions while ExpRate represents the percentage of correctly recognized expressions in handwritten mathematical expressions.
The loss function is defined as:

$$\mathcal{L} = \mathcal{L}_{cls} + \mathcal{L}_{counting}$$

where $\mathcal{L}_{cls}$ is cross entropy classification loss and $\mathcal{L}_{counting}$ is a smooth $L1$ regression counting loss defined as:

$$\mathcal{L}_{counting} = smooth_{L1}(\mathcal{V}, \hat{\mathcal{V}})$$

4.3 Result of Replication

The result is shown in tensorboard which includes evaluating ExpRate and WordRate, training Exprate and WordRate and loss in training and evaluating. The evaluating ExpRate, WordRate and training ExpRate, WordRate are shown from Figure 4 to Figure 5. After 175 epochs, the evaluating ExpRate reaches 0.5385 and the evaluating WordRate reaches 0.9037. Compared with the evaluating ExpRate in the paper on CROHME 2014, 2016 and 2019 with each ExpRate up to 57.00, 56.06 and 54.88, the result of replication is very close to ExpRate in the paper even though I only ran 175 epochs while author ran 240 epochs. Meanwhile, the evaluating ExpRate has already surpassed most of the current HMER method such as BTTR, DWAP-MSA and MAN, which further proves its SOTA ability of HMER among various current methods.

5 Improvement of CAN

5.1 Overview

Despite the significant performance improvement brought by the newly proposed model CAN, there are still some insufficiencies especially when encountering various writting styles or when extreme fine structure perception ability is needed. Besides, in order to get better performance, it's vital to further improve the accuracy of counting result and lower the counting loss. Meanwhile, the feature map extracted by the backbone network contains the most valuable information, so it's also important to improve the backbone network to extract more valuable information. Additionally, the training speed is extremely low. A new method is needed to boost the speed of CAN model.
Therefore, in order to conquer the above-mentioned drawbacks of CAN model, I tried some possible improvements on its backbone network and MSCM respectively.

5.2 Improvement on Backbone Network

The improvement I tried on the backbone Network is to replace the original DenseNet to a modified ResNet50 network.
DenseNet is a densely connected convolutional network, in which each layer is directly connected to every other layer in a feed-forward fashion. This means that, for each layer, the feature maps of all preceding layers are used as inputs, and its own feature maps are used as inputs into all subsequent layers. A DenseNet is composed of several dense blocks, where each block contains multiple convolutional layers. Between these dense blocks, there are transition layers which perform down-sampling using convolution and pooling operations to reduce the spatial dimensions of the feature maps. An example of DenseNet is shown in Figure 6.

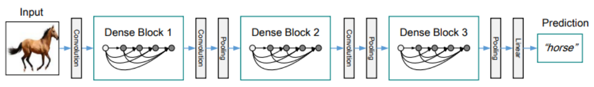

Figure 6: A DenseNet with 3 dense blocks

The advantages of DenseNet is clear that it can make the most use of feature map, thus capturing more valuable features in images. The parameters in DenseNet are also fewer than ResNet. However, DenseNet requires lot's of time on calculation due to its special structure and proposes a very high demand on GPU RAM to store feature maps.
ResNet50 is a residual network with 50 layers featuring for its residual structure. The advantages of ResNet50 includes its fast calculation speed and convergence speed. Due to the introduction of residual structure, ResNet mitigates vanishing gradient problem. Compared with densenet, less GPU RAM is needed but ResNet may introduces far more parameters as shown in Table 1. To obtain

Network	**Params**
DenseNet	2,979,288
ResNet50	14,841,452

Table 1: Params in two networks

the feature map $\mathcal{F} \in R^{H\times W\times 684}$, the last three layers : average pooling layer, 1000-d fc layer and softmax layer are removed from the original ResNet50. Additionally, a convolution layer is added to shift the channel from 1024 to 684, which corresponds to the channel of feature map in the paper.
I trained the two networks for 60 epochs respectively and kept tracking the evaluating ExpRate. The result is shown in Figure 7. Note that the ExpRate at epoch n represents the best ExpRate from epoch 0 to epoch n. The running time for 60 epochs is shown in Figure 8. From the result we can see that the running time of CAN applying modified ResNet50 is 1/3 running time of CAN applying DenseNet. While the evaluating ExpRate of modified ResNet50 is lower than the original model after 60 epochs, which can't be neglected.
However, by comparing the result of training ExpRate and WordRate, we can find that the modified ResNet50 model performs much better than the original model as shown in Table 2. Therefore,

Network	**Eval ExpRate**	**Train ExpRate**
DenseNet	**0.4221**	0.6289
ResNet50	0.3844	**0.8009**

Table 2: Evaluating Result

overfitting may occurred in the modified model

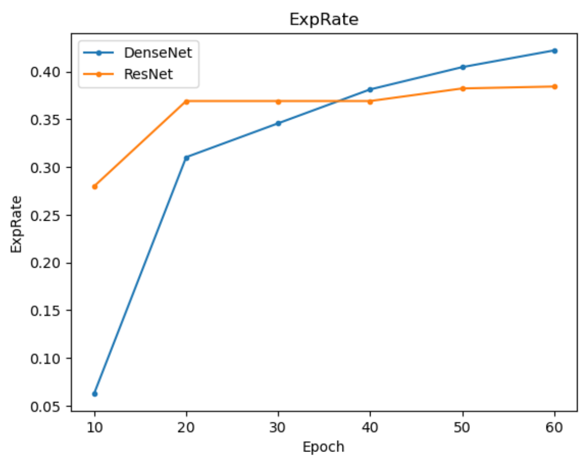

Figure 7: Evaluating ExpRate

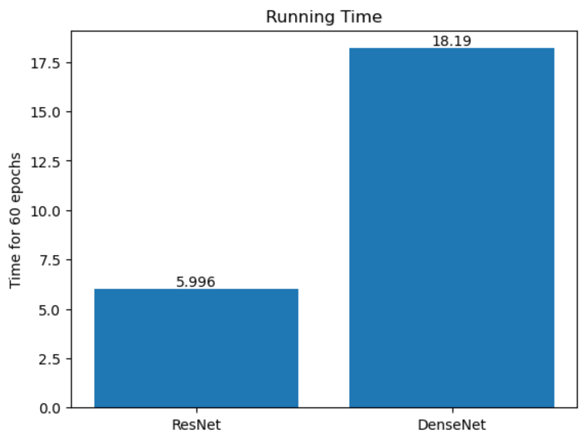

Figure 8: Running Time for 60 Epochs

since the params in the modified model is much larger than the params in the original model as shown in Table 1. If the overfitting could be solved by further modifying ResNet or changing some params, the performance of the modified model may be better while consuming less time.

5.3 Improvement on MSCM

As mentioned above, in order to boost the accuracy of counting result, MSCM should be first improved. A straight forward improvement is to add one more channel to extract multi-scale features.

Therefore, the improvement I made is to add a new convolution kernel of size 7×7 as shown in Figure 9. Combined with a larger kernel, it can better capture some large symbol such as $\sum, \lim, \ldots$, thus enhancing the accuracy of counting result.

Apart from the newly added kernel, I removed the

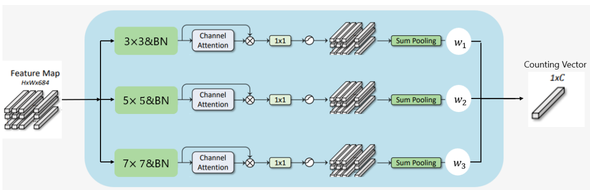

Figure 9: Structure of Modified MSCM

original average operator to combine the two counting vectors $\mathcal{V}^{3\times3}$ and $\mathcal{V}^{5\times5}$. Instead, three trainable weights w_1, w_2, w_3 are used to put the three counting vectors together:

$$\begin{aligned}\mathcal{V}^f =& w_1\mathcal{V}^{3\times3} + w_2\mathcal{V}^{5\times5} + w_3\mathcal{V}^{5\times5}, \\ s.t.\ & w_1 + w_2 + w_3 = 1\end{aligned}$$

By assigning different weight to different counting vector, MSCM can learn the importance of features extracted by different convolutional kernels and learn the optimal value that predict the most accurate outputs.

Similarly, I trained the CAN models with two different MSCM for 60 epochs and kept tracking the evaluating ExpRate. The result is shown in Figure 10. Teble 3 also shows the result. From the result we can clearly see the improvement brought by the new MSCM. At epoch 60, the evaluating ExpRate of the modified model is 0.02 higher than the evaluating ExpRate of the original model.

Network	**ExpRate**	**WordRate**
Original	0.4221	0.8402
Improved	**0.4350**	**0.8546**

Table 3: Evaluating Result

5.4 Further Improvement

Base on the improvement on MSCM, I further improved the MSCM and CCAD. With the aim of make the counting result more accurate and robust to different sizes of symbols, I split the MSCM

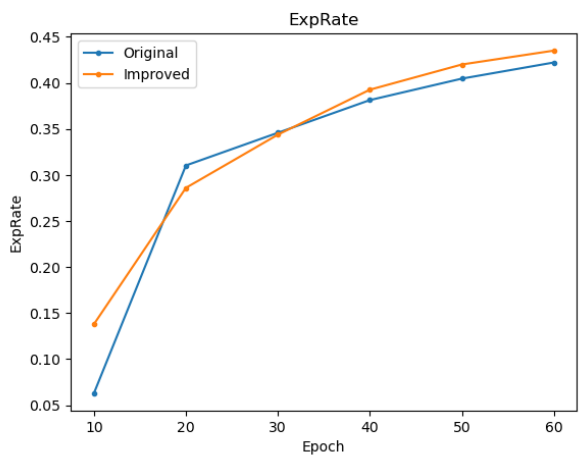

Figure 10: Evaluating ExpRate

into two parts, each consists of two convolutional kernels in different sizes as shown in Figure 11. In each part, the feature map passes two convo-

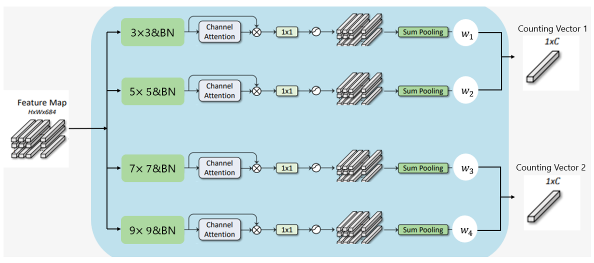

Figure 11: Structure of New Modified MSCM

lutional kernels in different sizes($3 \times 3, 5 \times 5$ and $7 \times 7, 9 \times 9$). The two extracted features are then enhanced to generate two counting vectors $\mathcal{V}^{3\times3}, \mathcal{V}^{5\times5}$ and $\mathcal{V}^{7\times7}, \mathcal{V}^{9\times9}$. The two counting vectors are than combined with four trainable weights w_1, w_2 and w_3, w_4 to get the final counting vector $\mathcal{V}^1, \mathcal{V}^2$:

$$\begin{aligned}
\mathcal{V}^1 &= w_1\mathcal{V}^{3\times3} + w_2\mathcal{V}^{5\times5}, \\
V^2 &= w_3\mathcal{V}^{7\times7} + w_4\mathcal{V}^{9\times9}, \\
s.t.\ w_1 &+ w_2 = 1 \\
w_3 &+ w_4 = 1
\end{aligned}$$

Finally, MSCM output two counting vectors. Note that the two parts utilizes four kernels: one with two small kernels and one with two large kernels so that the two output counting vector $\mathcal{V}^1, \mathcal{V}^2$ could each count small or large symbols with high accuracy.

Accordingly, I adjust the structure of CCAD to better utilize the two counting vectors as shown in Figure 12. Instead of simply add four counting

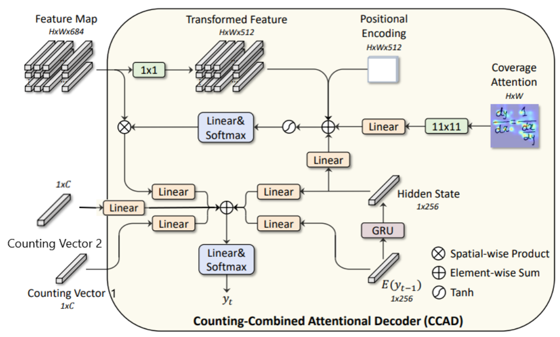

Figure 12: Structure of Modified CCAD

vectors together with w_1, w_2, w_3 and w_4, the new model first generates two counting vector $\mathcal{V}^1, \mathcal{V}^2$ in MSCM and each of them passes a Linear layer to generate the final prediction. The introduction of an additional Linear layer makes the most use of the two counting vectors which contains information of small/large symbols in high accuracy, so the accurate part of each counting vector could be exactly fetched by the Linear layer and used in generating outputs. The result of evaluating ExpRate is shown in Figure 13. The experiment compares

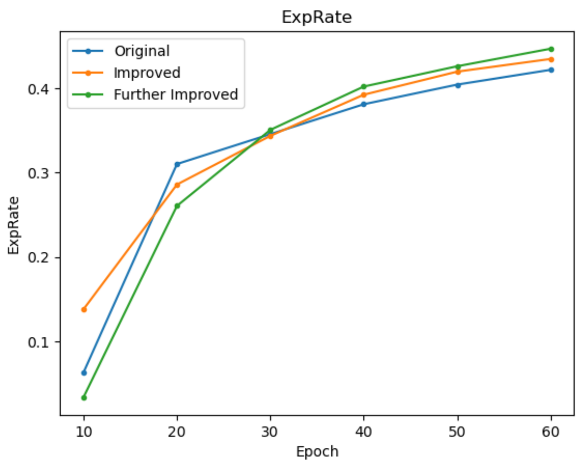

Figure 13: Evaluating ExpRate

three CAN models: the original model, the model proposed in 5.3(Improved) and the model proposed in this section(Further Improved). And the result shows that the further improved model performs best at epoch 60 over among three models. The evaluating ExpRate and WordRate at epoch 60 is shown in Table 4.

From the result, we can see that the further improved model achieves both the highest evaluating ExpRate and the highest evaluating WordRate.

Network	**ExpRate**	**WordRate**
Original	0.4221	0.8402
Improved	0.4350	0.8546
Further Improved	**0.4472**	**0.8605**

Table 4: Evaluating Result

6 Conclusion

In this project, I learn structure of CAN model and methods used in the paper. Besides, I replicate the algorithm in the paper and achieves similar evaluating ExpRate on CROHME datasets. Then I propose some improvements on its backbone, MSCM and CCAD. The results validates some conclusions. ResNet50 performs worse than DenseNet despite of its high speed. The introduction of another channel boost the accuracy of the model. The further improvement on MSCM and CCAD again enhances the performance of the model.

References

[1] Li B, Yuan Y, Liang D, et al. When counting meets HMER: counting-aware network for handwritten mathematical expression recognition[C]//European Conference on Computer Vision. Cham: Springer Nature Switzerland, 2022: 197-214.

[2] Huang G, Liu Z, Van Der Maaten L, et al. Densely connected convolutional networks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 4700-4708.

[3] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.