Report Draft

1. introduction and motivation

Solving handwritten math equations using typical gadgets like smart phones or computers is one the most desired ideas for reviewers and other researchers in general. The discovery of smart phone apps (e.g PhotoMath) or relevant other software modules have taken the passion to a reality. However, the apps and other modules, though impressive by some extent, there are limitations using these tools for handwritten math equation solving.

Handwritten math equations are hard to identify separately by machine and even when the components are identified it creates another complexity of determining the context of the whole equation. Therefore it is more complex of a situation to be dealt with in order to solve accurately. The adoption of neural network processing to identify math symbols from the input data and subsequently parsing it to a meaningful equation are making the whole process more obvious.

1. state of the art / related work,

(iii) your approach / design, implementation and evaluation,

To get a better overview of the whole process, we have ran both models that we found in the papers that are implemented by other authers in GitHub(https://github.com/jungomi/math-formula-recognition.git, <https://github.com/whywhs/Pytorch-Handwritten-Mathematical-Expression-Recognition.git>). We trained both models with crohme 2013 dataset, but after the training process, both models can't reach the theoretical performance that is posted in the paper. For this result, we have some hypothetical reasons for this bad performance. First, we are training both models on a small dataset(we have only 8835 items in the crohme 2013 dataset), it's difficult to get a well-performed model when we are lack of data to feed the model. Second, neither model is published by the writer of the papers. So there may be some small changes in the model, and there may be plenty of improvements that we can make. Since the Multi-Scale Attention with Dense Encoder model has a better theoretical performance than the WAP model, the model we're working on is based on Multi-Scale Attention with Dense Encoder model.

The original model form GitHub has an encoder-decoder structure. The encoder part contains 2 different inputs, one has the normal resolution so the dense net and convolutional neural network can extract the information of the structure and the symbols with bigger size, and the other has much higher resolution and it can extract the information about some details that may be neglected by in normal resolution. And the attention mechanism is applied in two different scales, by using attention, the model can focus on the informative annotations of the input images. In the decoder part, the model uses GRUs since the output is the latex form expressions, so the spatial relation plays a very important role here. GRUs can store long term memory which is very popular in some NLP tasks. And the task we are dealing with has some similar characters that are in common with NLP problems, in the math equation there is also relations between the former context and later context. The beam search is also implemented, the result is the one with the highest probability. The model has the input image size of 128\*128 pixels, and the output is a latex form equation, the tokens in the expression has 118 different types. It can recognize normal plus, minus, multiply, divide, fraction, integration, root, exponent equations. But it can’t recognize matrix calculation.

After understanding the model, we started to try to make the performance better. After a few tries, we got several models by applying different learning rate changing functions, the best model we got has the WER (Word Error Rate) of 40.93% and the Correct expressions rate of 1.13%, and it's still far from the theoretical performance (WER 12.9%, correct expressions rate 52.8%). So, we want to analyze the cause of this poor performance.

So intuitively we come up with 2 solutions. First is more training data, secondly is higher resolution. To justify those two solutions and find more potential factors that may affect the result, we make some statistical work to see which type of expression or token is more likely to be wrongly recognized. Besides, we also checked the wrongly recognized expressions to see if there is any feature that is in common. And the statistical analysis can also show a more accurate evaluation of the models. By comparing the statistical results of different models, we can find out the improvements and the differences. As a result, each time we trained a new model with different hyperparameters, different datasets or different model structure, we always check the WER and correct expressions rate as an intuitive index at first, and if it’s worth analyzing, we make the statistical analysis of the testing result.

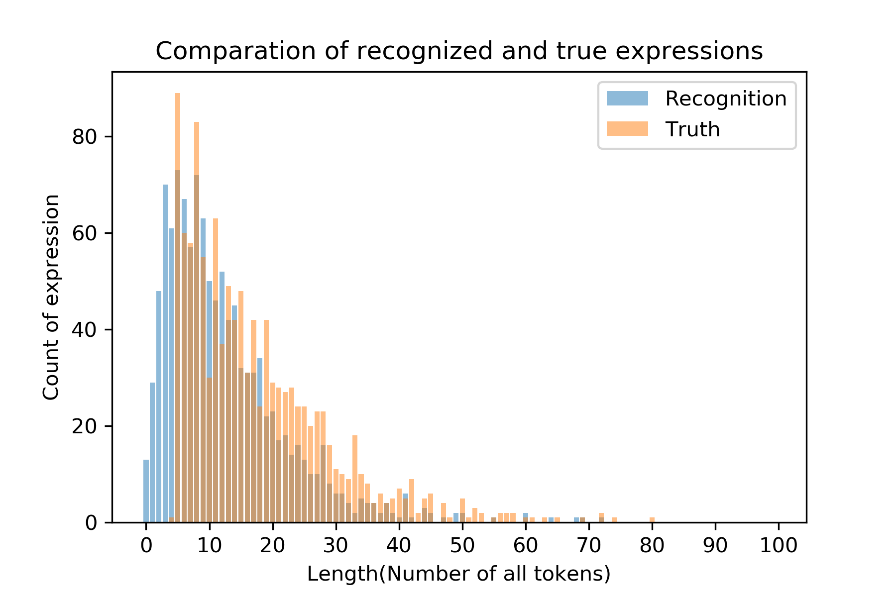
The statistical analysis is divided into two parts, one is the analysis on the expression level, it may have macro perspective of how the expressions are recognized and different recognition result of expressions with different length, and the other one is on the token level.

Expression analysis:

At first, we want to list the types of wrong recognitions.

1. Missing tokens, this type of wrong recognition means that the token that is in the actual expression, is not recognized at all, but missing in recognition.
2. Wrongly recognized tokens, this type of wrong recognition means that the existence of the token is recognized but the token is wrongly recognized (e.g. ‘m’ is recognized as ‘nn’ or ‘z’ is recognized as ‘2’) or its position is wrong (e.g. ‘god’ is recognized as ‘dog’).

We counted the length of the actual expressions, the length of the recognized expressions, and the Levenshtein distance between those corresponding expressions. And we counted these three distances in two different conditions: 1. full expression that includes all the spatial tokens, it can provide an overview of the error source 2. The symbol only, it can provide the error source of the symbol itself. By comparing these statistical

The pics below show the distribution of expressions with different lengths.

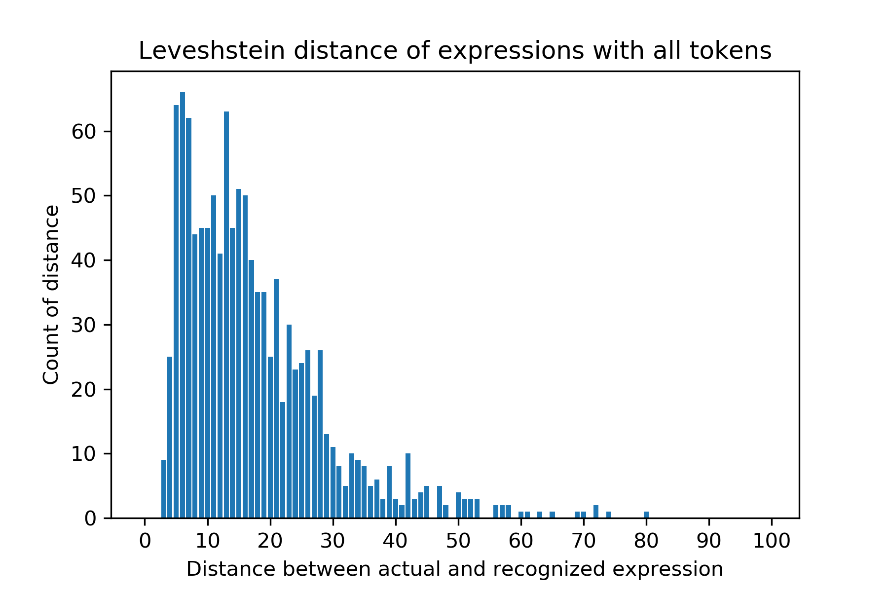
Figure1: Comparation of length distribution of recognition and truth

Figure2: The distribution of Levenshtein distances of every recognition and its truth

From the comparison of two distributions in figure.1, we can find out that the number of long expressions in the actual label is more than the number of long expressions in the recognized results. That means that the long expressions may be recognized as some shorter expressions. And the expression length distribution between 0 to 10 has also obvious difference. That could mean the images that contain small number of tokens may be also recognized wrongly. And the model is likely to recognize the expression as some other expression with less tokens, some tokens are missing.

The reason for this result maybe the resolution is too low so that some details are not recognized by the model. The possible solution for this problem is raising the resolution of input images. The input resolution of this model is 128\*128, we tried to raise the resolution four times bigger, which is 256\*256. And the data preprocessing is possibly needed, so that the image that contains the few tokens can be resized into normal size, or we can and another scale in the model, since the model now contains only 2 scales, which are normal resolution and high resolution. If we add a low-resolution scale in the model, it can extract the information better when it has a higher view of the images.

From the distribution of the Levenshtein distance in figure.2, we can draw the conclusion: most of the error length stays in the intervals of 0 to 20, but there still are many severe mistakes that have the Levenshtein distance that’s bigger than 20, which means a total false.

Tokens analysis:

Analysis of the error recognitions that have severe problems:

We checked the Figures that have big Levenshtein distance, if the distance is smaller than 5, we just consider it as some acceptable error. After sampling from the error recognitions, we found some common features of the misrecognized expressions: either the formula has some complicated spatial structure, or the formula is very long, that with the resolution of 128\*128 it’s even hard for a human to distinguish the symbols in the image. And some other kind of errors often occurs, one important type is similar tokens, like ‘z’ and ‘2’, uppercase tokens and lowercase tokens, etc.

With the hypotheses above we tried the model with higher resolution (256\*256) and more data, the outcome is better than the previous model. This model has the WER of 35.41% and the correct expression rate of 10.29%.

To evaluate the outcome of the new model, we also did some statistical analysis on the testing result, in expression level and token level.

Expression analysis:

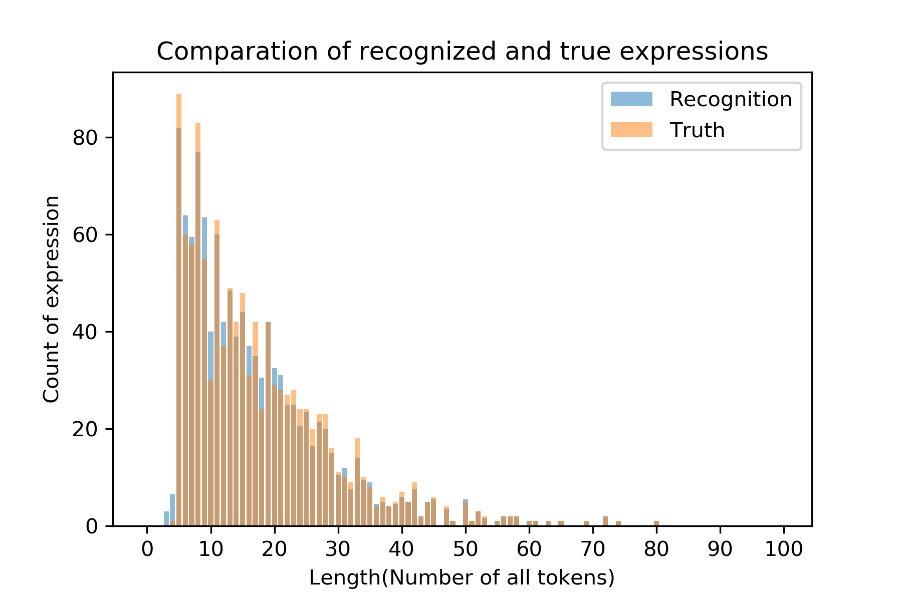


Figure3: Comparation of length distribution of recognition and truth (higher resolution and more data)

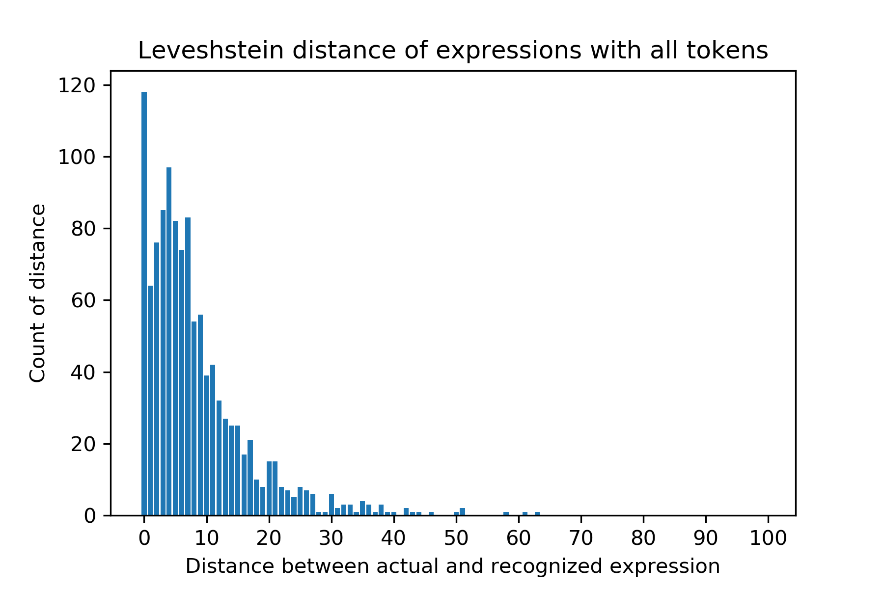


Figure4: The distribution of Levenshtein distances of every recognition and its truth (higher resolution and more data)

In the figure 3 we can see some very obvious improvements of the performance. The problems

we found in the low-resolution model are prominently relieved, the missing token problem in previous model is basically solved, since the length distribution of the recognized expressions is mostly overlapping with the length distribution of the actual expressions, the distributions of expression lengths in truth and in recognition are similar. The recognized expressions of length from 0 to 4 are less than the results from previous model, and the number of the expressions with more tokens are alike in truth and recognition.

*With this observation we can draw some conclusions that the details in the images are mostly captured by the model, and the model can notice the existence of most tokens. The main problem of the misrecognition is not missing tokens but wrongly recognized tokens.*

In figure 4, we recalculated the Levenshtein distance of each expression and its corresponding recognized expression. The distribution of the Levenshtein distance has 2 spikes, one is the 0, which means that there is no difference between truth and recognition and the recognition is totally correct, and another spike is at 4, which means that most (80%) wrongly recognized expressions lies in interval [1,12] and have the peak at 4.

To solve the wrongly recognized token problem, we think one better solution is add more CNN(convolutional neural network) layers in the models, when the model gets deeper it can acquire better learning abilities about recognizing the tokens.

The fig. shows the model structure in the paper (Multi-Scale Attention with Dense Encoder for Handwritten Mathematical Expression Recognition), and there is only one CNN layer at the beginning of the model (which has kernel size of 7 channel of 48 and stride of 2) except the CNNs in the dense net. This is not enough because only 1 layer of CNN can only extract some simple patterns, but the tokens in the handwritten equations have some spatial relations. For example, the subscript contains the spatial information between 2 tokens and this type of information can’t be extracted by a single layer of CNN. To improve the ability of the model to recognize the patterns more accurately, we replaced this 1 layer of CNN as 3 layers of CNNs, so the model can feed information about more complicated patterns into the encoder.

After training on the same dataset, and testing on the same 2016 testing set, the result of the model gets better. The WER gets 33.05% and the correct expressions rate gets to 12.21%. For this result, we did the same statistical work as the last two model.

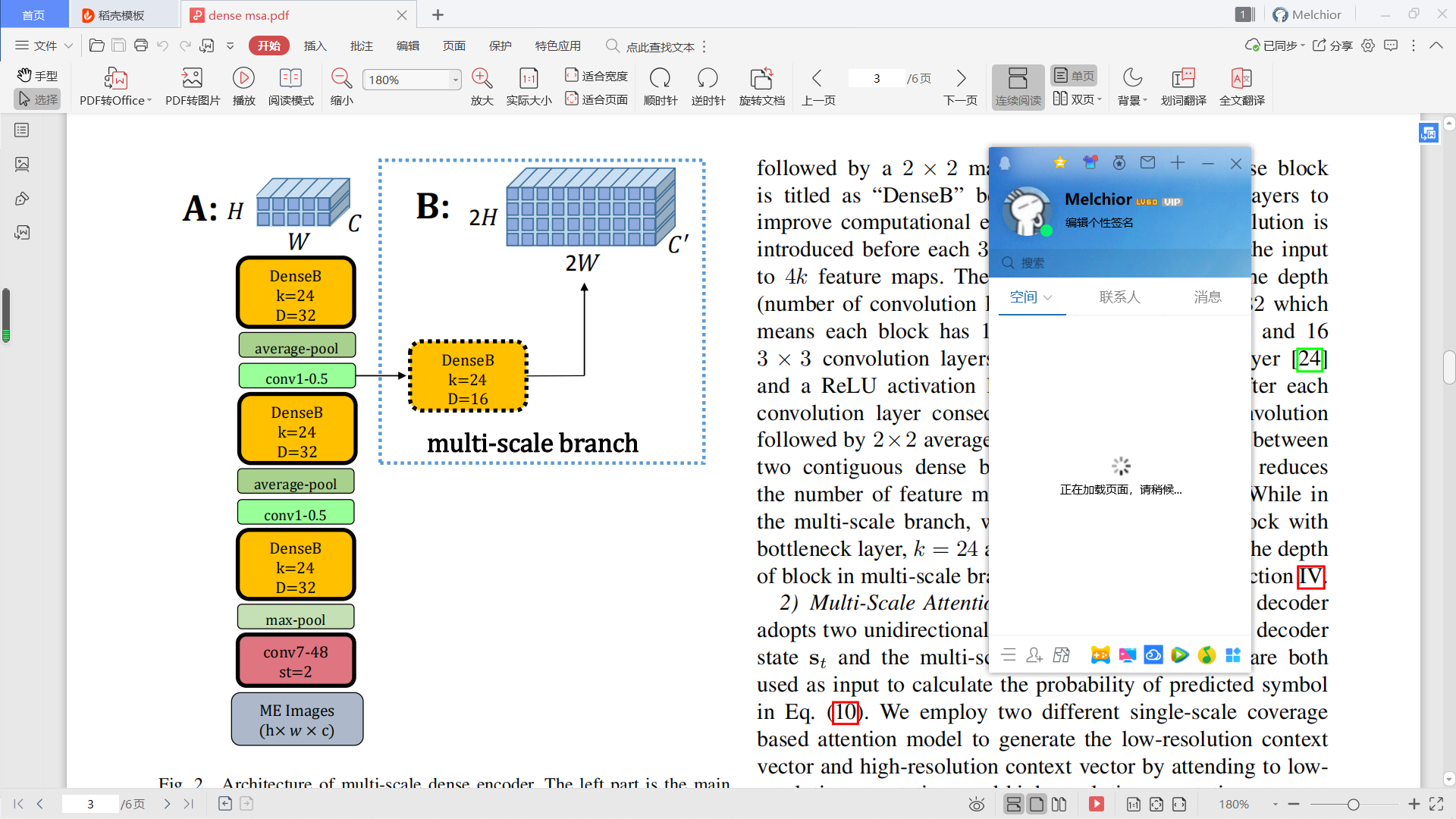


Figure.

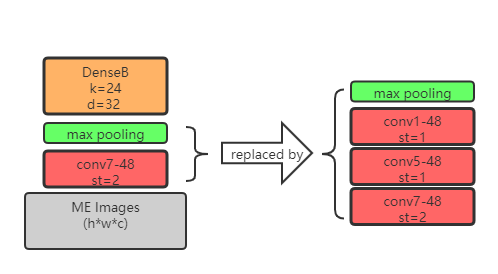


Figure.

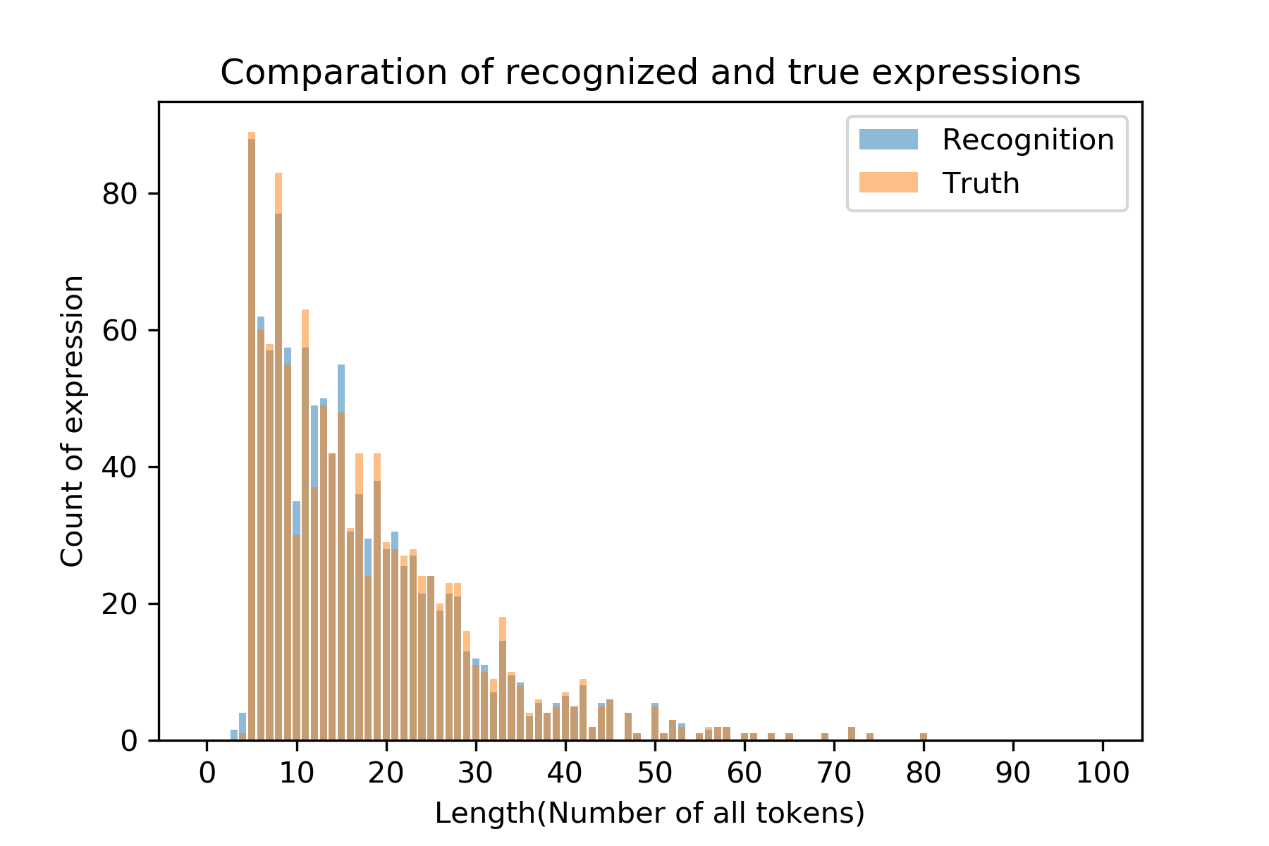


Figure.

From the comparation in figure. we can see the distributions of expressions in recognition and truth have some changes in compare of the comparation in last model, but the changes is not very obvious.

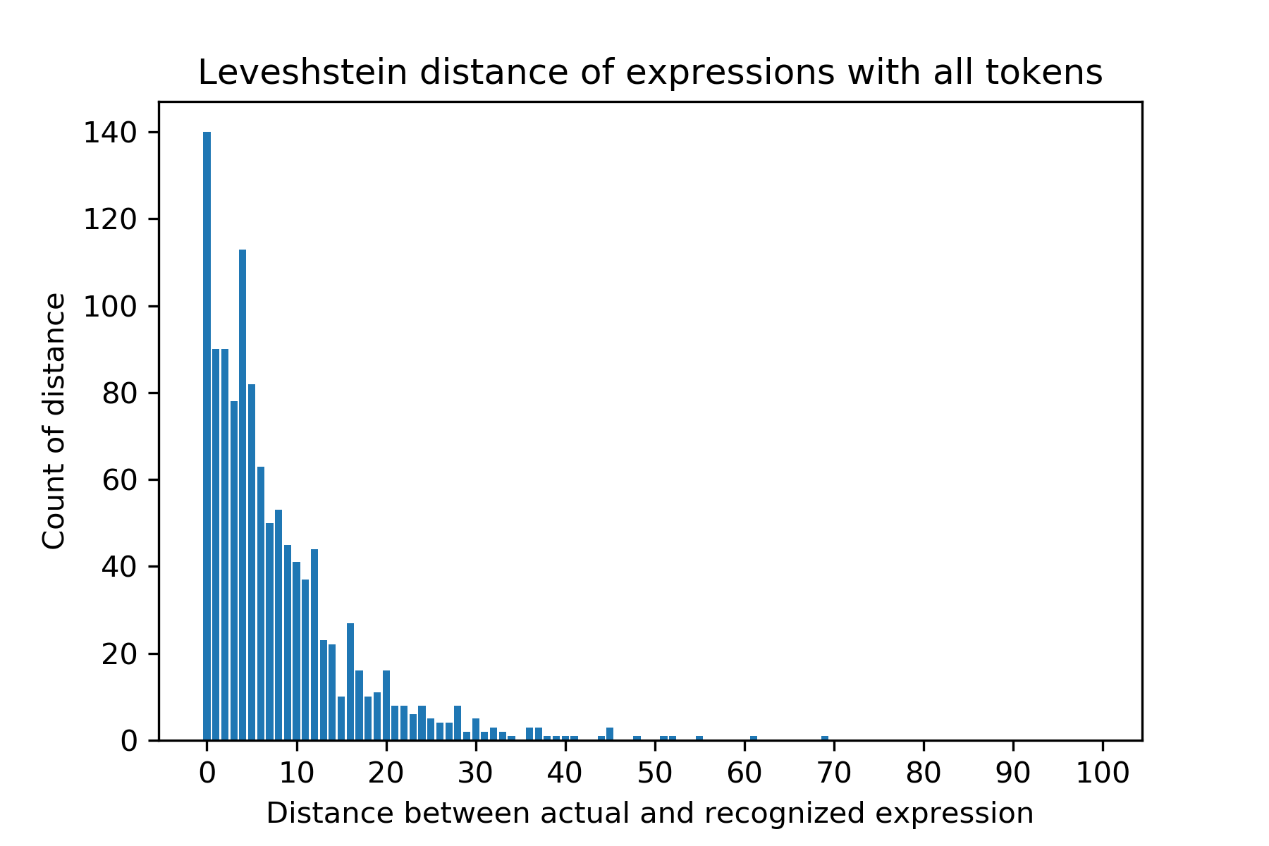


Figure.

But the distribution of the Levenshtein distance in figure. has some significant changes in compare of the last model. The counting number of expressions which have 0 distance is higher than the last model,which means the correct recognized expressions are more. And the most (80%) wrongly recognized expressions lies in interval [1,12]///??????????????????? . This statistical result means that the wrongly recognized tokens is less than the last model. By adding more CNN layers has positive effects on the problem we found.

iv discussion and conclusion