Fernando Vallecillos Ruiz #1W18CG03-8

2019 PRMU challenge on old Japanese character recognition

Introduccion

This challenge consists of recognizing three successive characters in old Japanese documents, and output the Unicode of the set of three characters. We know that every sample will only have 3 characters. Furthermore, these characters belong to the set of hiragana (no katakana or kanji). We also have 3 different datasets:

- Training set: 119.996 elements
- Test set: 16.387 elements
- Kana set: 388.146 elements

The training set and the test set contains samples with 3 images. On the other hand, the Kana set contains samples with only 1 character. Since we do not have the answers to the test set, we would be splitting the sets into 80% training and 20% test sets. These would be selected randomly.

In this final report I will describe my process and the technical details of my solutions to solve the 2019PRMU challenge.

Preprocessing

The first step is to preprocess the images. We have different number of problems that we need to take care before even trying to use any of the samples. If you take a look at the samples there are 3 main problems that a lot of the samples have in common.

- Color: we do not need colors for the images. It does not matter the color of the background for the identification of the characters.
- Ink: since we are dealing with characters written in ink there are two main problems. Some of the traces are very subtle. In addition, the ink fades away from the paper, making some of the traces have white spaces.
- Transparency: again, since we are dealing with characters written in ink over paper, some of the characters from the other side of the paper tend to show through, making very defined gray traces.

		
Color	Ink	Transparency

For these problems there are three different solutions. These solutions can be translated to different libraries in Python. The main ones being CV2 and Image. I finally chose Image because the final results were better (personal point of view) and the processing was a little bit faster:

- Color: this is the simplest solution. We only need to transform the image to grayscale.
- Ink: we are going to use morphological transformations. These are simple operations based on image shape. In CV2 it would be the ‘erode’ function. In Image I found that ‘RankFilter’ works the best. It creates a rank filter (sorts pixels in a window and returns the rank’th value).
- Transparency: I increased the contrast of the Image. After the other filters, it will increase the contrast of the main letters making it stand out over the background noise.

Finally we would do a resizing process. The samples’ sizes vary too much to get an average in a simple way. Because of these I made a simple script to get the average width and height of the training and kana set.

- Training set:
    - Average width: 92,87 pixels
    - Average height: 282,37 pixels
- Kana set:
    - Average width: 72,17 pixels
    - Average height: 89,78 pixels

With this data I tried to maintain the proportion and tried different approaches with 2 different sizes:

- Width: 72 pixels. Height: 180 pixels (60 for kana)
- Width: 40 pixels. Height: 150 pixels (50 for kana)

The smaller size seemed to work better in the machine learning approach and it performed very closely in the algorithmic approach. Accounting with the difference of total of pixels to process (less than half), I decided to use the second size.

Methods

Since this was my first approach to a computer vision problem, I wanted to try different approaches. By doing a little bit of research about the best way to tackle this problem, I came around CNNs. I wanted to try them but there was no certainty I could make it work. Therefore, I also tried an algorithmic approach. What it's more, I also wanted to try these two approaches with the two different sets. Because of this, I will be getting 4 sets of results.

Algorithmic Approach

I was fairly sure that the machine learning approach with CNNs would obtain better results. However, for the sake of learning and experimenting, I tried to come up with an algorithmic approach which is completely clear, in contrast with the 'black-box' style, typical of the neural networks.

The algorithm can be summarized in a really simple way:

1) Preprocess the sample (discussed before)
2) Resize the sample
3) Create a feature space based solely on the value of the pixels.
4) Evaluate the image:
    a) Preprocess test image
    b) Resize the test image
    c) Subtract the image from every image in the space
    d) Calculate Frobenius Norm for each one
    e) Select the lowest result

It may be a little bit hard to grasp, so I will try to explain it in a little bit more in detail. Before step 4), we have a feature space with every image. We're are only saving the values of the pixels, nothing more. Then, after preprocessing and resizing the test image, we subtract the test image from all the images. Our space now is formed by the difference of every image with the test image. Therefore we have values ranging [-255, 255]. This is when the Frobenius Norm is applied. The frobenius Norm is also called the Euclidean norm, and is defined as the square root of the sum of the absolute squares of its elements:

$$||A||_F = \sqrt{\sum_{i=i}^{m} \sum_{j=1}^{n} \left|a_{ij}\right|^2}$$

By applying the Frobenius norm to each difference, we would accumulate in one number the difference in the values of pixels for each image. After that we would only need to choose the one with the lowest value (the most similar one).
We can see that this is a very naive approach, however it is still an algorithmic one that was very useful to get comfortable with the different tools as well as an introduction to the problem.

Machine Learning Approach

As mentioned before, I used a convolutional neural network in this approach. Personally, I didn't know what they were or how to use them but, for the sake of not extending this paper too much, I will assume that the reader know what they are.
Since it was my first approach to this, I modified the parameters many times and trained many different models (46 in total). I tried with different resizing parameters, as well as, different number of dense layers, convolutional layers, neurons in dense layers and neurons in convolutional layers.

Results of algorithmic method

I will briefly explain my first try of this method with the training set. I would try to map around 120.000 samples in the feature space and then use it to evaluate new images. It could be seen like a foolish approach, however, there is at least some possibility that it could work.
We can do some math to prove it:

$$_{48}P_3 = P(48, 3) = 103.776$$

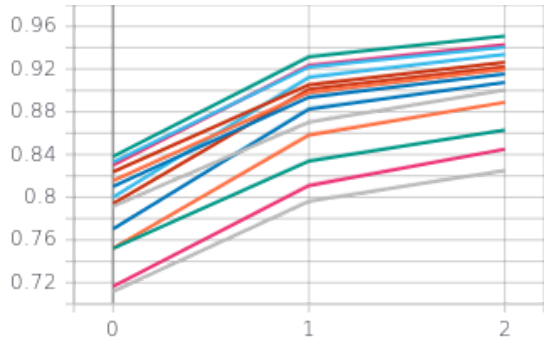
By using permutations, we know that there are in total 103.776 different possible combinations with the 48 kanas in the set. Since our training set it's composed of around 120.000 elements, it is theoretically possible, to map every possible combination in the space.
However, it was computational infeasible for me. I do only have a laptop to do everything and it doesn't have enough memory for this to work. However, it could be interesting to see how far we can get with this.
In the light of these results, I couldn't hope to use the kana set (almost 3 times as big).
Nevertheless, I tried a new approach for this. I tried to first split samples of the training set into three pieces, and then map them. Since the chunks of memory required for this were a third of the original size, my computer would have enough space in memory to allocate them. I found that I could get away with this using half of the training set.
Then, a different problem appeared, the memory was enough, but the processing capabilities were not. For half of the training set (60.000 elements), it would take at least 20 seconds to evaluate each image into the space. Because of this I took a rather smaller test set, 600 elements, which took around 3 and a half hours to process. To my surprise, I got an accuracy of almost 50%. Way more than I expected for this kind of naive approach.

Results of machine learning methods

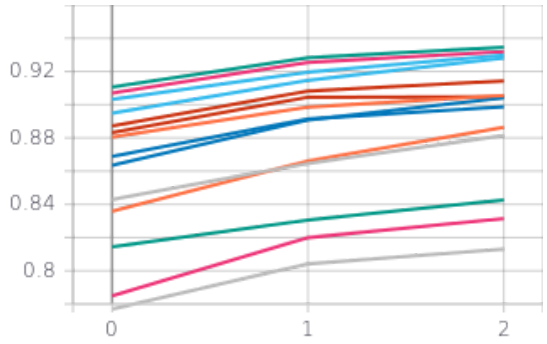
For this part I used the Keras library in Python. I ran it with tensorflow on the background and using the GPU to parallelize the training.
First of all I used the Pickle library in Python, which is used for serializing and de-serializing a Python object structures. With this I was able to create the different datasets and later load them many times in the CNN. Because of the way that Keras works, I also used a one hot encoding for the different labels. I also normalized the values of the samples to make them in the range [0, 1].
With this, everything was ready to start training different models. I started trying different parameters that I read could be good and from there making changes. Of course, I was limited by my laptop in terms of memory, processing capabilities and time (I couldn't use it while it was training). Nonetheless, I tried many different combinations and the results were good. I must also mention that we used the Adam Optimization Algorithm for deep learning (I read it was the most appropriate) and Cross-Entropy for my loss function.

First, I will start analyzing the training with the Kana set. We had almost 400.000 samples so it was big enough to not make me worry about overfitting however, there was a really big difference in the number of samples from the different kana sets. With this I want to say that there were sets with as many as 24.000 elements, meanwhile, some other sets had as little as 400 elements. I tried to use weights to balance this, but the imbalance was soy big that it could have affected the final results.

For the Kana set we obtained pretty promising results:

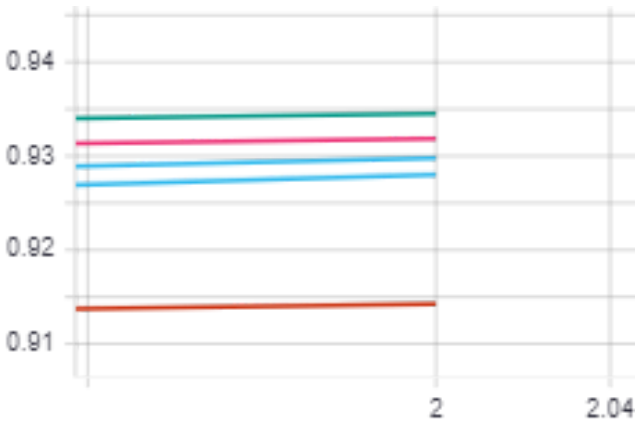


Accuracy in the training set



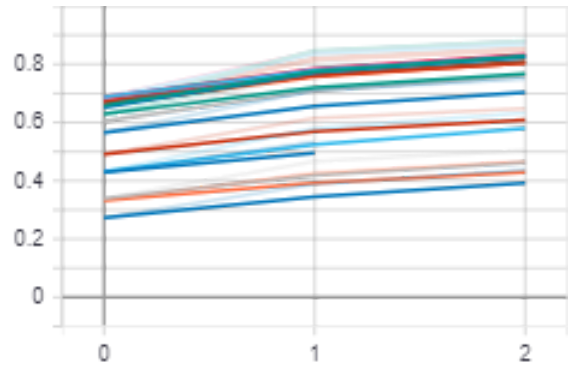
Accuracy in the test set

As we can see, the accuracy in the training set got to almost 0.96, increasing more than 0.1 in the first epoch. Of course, the increase in test set is more subtle.
We can go ahead an analyze this more closely.

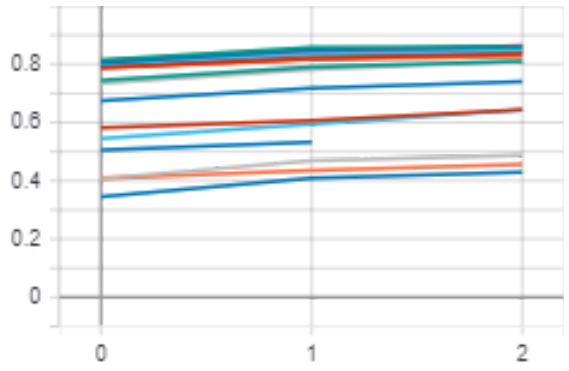


Name	Smoothed	Value	Step	Time	Relative
128-conv_nodes-3-conv-128-nodes-1-dense-1562354825--CNN	0.9298	0.9299	2	Sat Jul 6, 05:17:35	33m 37s
256-conv_nodes-3-conv-128-nodes-1-dense-1562357856--CNN	0.9318	0.9319	2	Sat Jul 6, 06:24:47	44m 45s
256-conv_nodes-3-conv-64-nodes-1-dense-1562318228--CNN	0.9143	0.9144	2	Fri Jul 5, 19:25:28	45m 31s
512-conv_nodes-3-conv-128-nodes-1-dense-1562361888--CNN	0.9345	0.9346	2	Sat Jul 6, 08:12:22	1h 11m 35s
512-conv_nodes-3-conv-64-nodes-1-dense-1562322328--CNN	0.928	0.9282	2	Fri Jul 5, 21:14:21	1h 12m 16s

Taking a closer look, we see that the best results increase with additional convolutional layers. Even if the accuracy in the training set was good, additional dense layers tend to overfit the set (less accuracy in the test set). We also can see how the number of nodes in the layers do not necessarily produce a better result. Nonetheless, it makes the process go slower (more than twice the time).
Now we go to the final results, training the CNN with the training set. The set is smaller compared to the last one. Also, since these images are closer to what an actual sample would look like, it is a more realistic approach to the problem. These are the results:

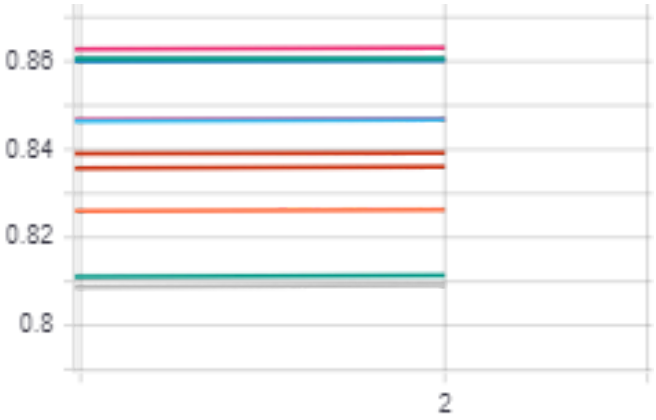


Accuracy in the training set



Accuracy in the test set

Right away, we can appreciate how the curve is not as abrupt in the training set, an almost non existent in the test set. We can also guess that the distribution of the density of the samples is, at least, not as imbalanced as in the Kana set. Let's take a closer look and analyze the results:



Name	Smoothed	Value	Step	Time	Relative
THREE--1024-conv_nodes-4-conv-128-nodes-2-dense-1562541356--CNN	0.8607	0.8608	2	Mon Jul 8, 17:29:53	5h 55m 0s
THREE--1024-conv_nodes-4-conv-128-nodes-2-dense-1562642454--CNN	0.8602	0.8604	2	Tue Jul 9, 19:18:16	4h 41m 21s
THREE--128-conv_nodes-3-conv-128-nodes-1-dense-1562418053--CNN	0.836	0.8362	2	Sat Jul 6, 23:11:24	46m 55s
THREE--256-conv_nodes-3-conv-128-nodes-1-dense-1562422284--CNN	0.8467	0.8469	2	Sun Jul 7, 00:43:25	1h 1m 20s
THREE--512-conv_nodes-3-conv-128-nodes-1-dense-1562427805--CNN	0.8468	0.8469	2	Sun Jul 7, 03:05:59	1h 35m 4s
THREE--512-conv_nodes-3-conv-128-nodes-2-dense-1562518195--CNN	0.8392	0.8393	2	Mon Jul 8, 04:12:59	1h 35m 30s
THREE--512-conv_nodes-4-conv-128-nodes-2-dense-1562529419--CNN	0.8631	0.8634	2	Mon Jul 8, 08:15:55	2h 20m 49s

Again, the performance increases with the number of convolutional layers, however, there is not that much difference between 3 and 4 convolutional layers. We can see that additional dense layers don't seem to overfit as much, maybe because of the extra convolutional layers. What increased surprisingly is the difference in time by changing some parameters. In these results we can see that, with a small difference in the final result, the time that it took from training is 8 times bigger.

Conclusion

In this work, I have introduced two possible solutions for the 2019 PRMU challenge. I did not conform with only one solution but tried to have two radically different approaches, algorithmic and machine learning. Although the naivety of the algorithmic solutions, I have shown that they are theoretically possible, as well as perform 'decently'. On the other hand, the machine learning solutions uses a more high level and advanced approach, therefore getting better results. The analysis of the many models show the different progression and accuracy depending on the parameters of the CNN.

Sadly, the PRMU competition in codalab has been down for almost two weeks therefore, I have no way to test the results in the test set provided. Hopefully, they it will be fixed before the end of the competition so the actual results can be checked.

The code for the results will be attached with paper. I will also attach the folder 'logs' with the needed files to run tensorboard, a tool to show accuracy and lost in the training set and validation set for every one of the 48 trained models.

References

http://cs231n.stanford.edu/reports/2016/pdfs/262_Report.pdf

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

https://pillow.readthedocs.io/en/5.1.x/reference/ImageFilter.html

https://docs.opencv.org/trunk/d9/d61/tutorial_py_morphological_ops.html

http://ishidate.my.coocan.jp/vpy_8/vpy_8.htm

https://github.com/RakuTheSenpai/Hiragana-Identifier/blob/master/src/cnn.py

https://towardsdatascience.com/a-simple-cnn-multi-image-classifier-31c463324fa

https://medium.com/@contactsunny/label-encoder-vs-one-hot-encoder-in-machine-learning-3fc273365621

https://www.kaggle.com/pierrek20/multiclass-iris-prediction-with-tensorflow-keras

https://dev.to/hydroweaver/single-label-multiclass-classification-using-keras-45jl

https://towardsdatascience.com/handwriting-recognition-using-tensorflow-and-keras-819b36148fe5

https://towardsdatascience.com/introduction-to-deep-learning-with-keras-17c09e4f0eb2

https://realpython.com/python-keras-text-classification/

https://machinelearningmastery.com/how-to-make-classification-and-regression-predictions-for-deep-learning-models-in-keras/

https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

https://www.d2l.ai/chapter_convolutional-neural-networks/why-conv.html

https://pythontips.com/2013/08/02/what-is-pickle-in-python/