MATHEMATICAL SYMBOL IDENTIFICATION: A CNN IMPLEMENTATION

CSCI-SHU 360 MACHINE LEARNING FINAL REPORT

Chengxun, Wu
NYU Shanghai
cw2961@nyu.edu

Yuxuan, Li
NYU Shanghai
yl5582@nyu.edu

Xingjian, Gao
NYU Shanghai
xg760@nyu.edu

December 2020

ABSTRACT

Convolutional Neural Network (CNN) is one of the most robust tools in dealing with images in machine learning and deep learning tasks. This project focuses on the implementation of hand-written mathematical symbol classification based on Convolutional Neural Networks. After processing the original images into tensor-based data, this project constructs the CNN structure based on *pytorch* and improves the model using different optimizers. After 25 epochs of training, the CNN-based classifier obtained a performance with a test accuracy of 98.7% and is successful in identifying personally-written samples.

1 Introduction

The inspiration of this problem comes from a mathematics student's daily life: how to electronically store the hand-written mathematical formulas and expressions into the standardized digital form? Typing on Latex would be clearer while writing it on the paper would be much faster but with less clarity. This trade off inspired the team with this Mathematical symbol identification idea. Technically speaking, this is a supervised classification problem. Famous example of this would be the identification of numbers of 0-9 using Convolutional networks (CNN) which the team starts with. In the paper *Gradient-Based Learning Applied to Document Recognition*, Professor LeCun elucidates the robustness of CNN by highlighting that "Multi-layer Neural Networks trained with back-propagation algorithm constitutes the best example of a successful Gradient-Based Learning technique" (1). Compared with the original digits recognition, this project corresponds to a multi-class classification problem, which deals with a more complex and practical task.

2 Data Preprocessing

2.1 Data Overview

The project utilizes data downloaded from Kaggle[1] with an original dataset containing $385,459$ pictures of size 45×45 pixels from 82 different categories. The categories contain the labels of numerical digits (e.g. 0$\sim$9), alphabets, Greek alphabets (e.g. α, β) and other mathematical symbols (e.g. ∞, $\int$ and $\forall$).

Since the dataset is comprehensive with computer-friendly sizes, so despite the distribution of occurrences of different categories is slightly unbalanced, the team decided to utilize all of the data to the training, testing and validation processes. In order to better store the information of categories in numeric form, the project uses a function to translate between index and category names. The distribution of the original data can be found in figure 1. Based on the data, the team starts processing the data into tensors and store them for further applications.

[1]Link to the data from Kaggle: https://www.kaggle.com/xainano/handwrittenmathsymbols

2.2 Data Processing

Firstly, the team splits the pictures into train, validation and test folders with respective proportion $60\%, 20\%, 20\%$. After the random splitting, each folder has data of size in table 2. The team uses packages from *torchvision* to convert the pixel-based pictures into tensors then stores the pictures. The team first stores the data in RGB format, converting each picture to tuples in the form of

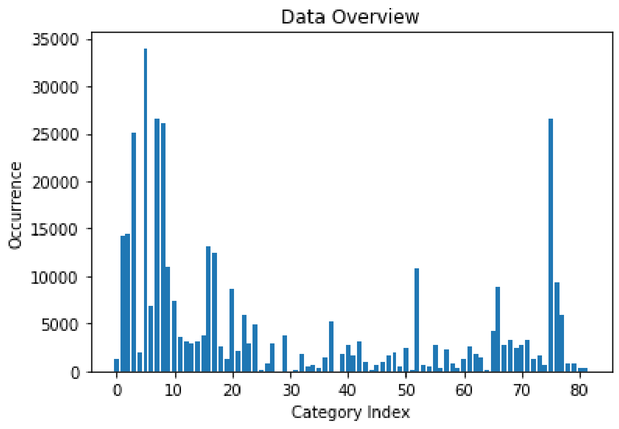


Figure 1: Data Overview

Dataset Type	Size
Train	225552
Validation	75261
Test	75161

Figure 2: Sizes of data after splitting

$$(\text{tensor}(\text{size} : [1, 3, 45, 45]), \text{index of label})$$

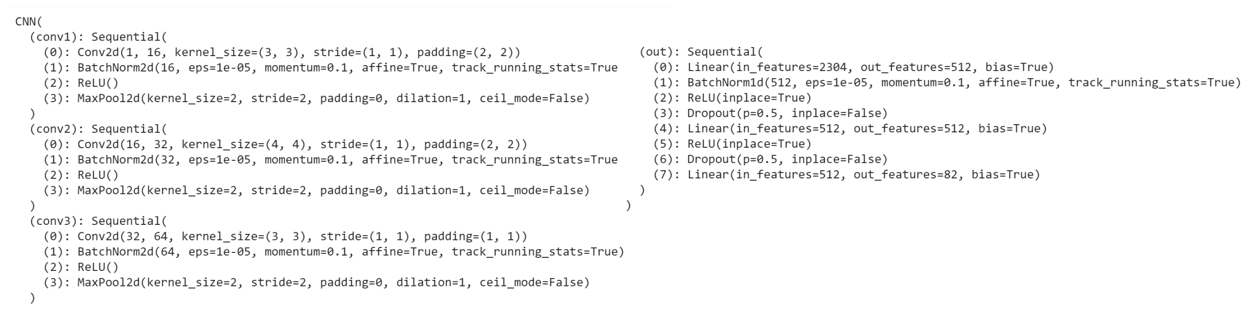
However, after updating the data into *Google Drive*, the loading process in *Colab*[2] would terminate due to insufficient space in RAM and GPU. Hence, the team converts each picture into tensor with size $[1, 1, 45, 45]$ after noticing that the dataset only contains black-and-white images. Such conversions would not jeopardize pictures' information and the team witnesses a considerable reduction in the cost of space.

Moreover, the setting of the 4-th dimension in the tensors allows batch-based data loading in training and testing, which accelerates the training and testing processes. After creating different datasets, to avoid the unnecessary time cost of data processing in *Colab*, the datasets are stored locally and uploaded to *Google Drive* for subsequent implementation.

3 Methodology and Algorithms

3.1 Convolutional Neural Network Structure

The primary structure the project applies is a CNN model. The team decides on a structure with three convolutional layers and three fully connected layers. The network structure is given by figure 3, which is an overview function offered by *pytorch*. Tools online allow the team to present a simple visualization[3] of the constructed CNN (See Figure 4).



```
CNN(
  (conv1): Sequential(
    (0): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(16, 32, kernel_size=(4, 4), stride=(1, 1), padding=(2, 2))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv3): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
```

```
  (out): Sequential(
    (0): Linear(in_features=2304, out_features=512, bias=True)
    (1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=512, out_features=512, bias=True)
    (5): ReLU(inplace=True)
    (6): Dropout(p=0.5, inplace=False)
    (7): Linear(in_features=512, out_features=82, bias=True)
  )
)
```

Figure 3: CNN Architecture

[2]Thanks to https://colab.research.google.com/.

It leaves out the details (e.g. Batch Normalization, Drop-out mechanism), only leaving the skeleton of the construction: convolutional layers, max-pooling layers and fully connected layers.

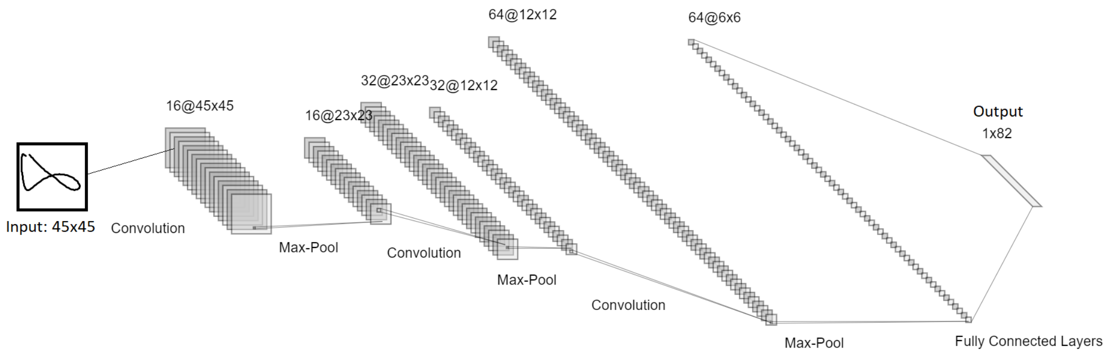


Figure 4: Simplified Visualization of CNN

3.2 Model Evaluation Metrics

As a classification problem, the project uses the Cross Entropy Loss function as the function for measuring losses. The Cross Entropy Loss function offer by *torch.nn* works as follows[4]:

$$\mathrm{loss}(\mathrm{x}, \mathrm{class}) = -\log\left(\frac{\exp(\mathrm{x}[\mathrm{class}])}{\sum_{\mathrm{j}} \exp(\mathrm{x}[\mathrm{j}])}\right) = -\mathrm{x}[\mathrm{class}] + \log\left(\sum_{\mathrm{j}} \exp(\mathrm{x}[\mathrm{j}])\right)$$

where x is the input data (output for each class) and class refers to a specific class to measure the loss. It is a combination of soft-max function and traditional cross entropy function. Also, it works well for data with a higher dimension, allowing batch-processing, hence the team decides to apply it directly to the output of the constructed CNN.

For the model evaluation, we decided to use prediction accuracy as the yardstick, which is defined as the following:

$$\mathrm{Accuracy} = \frac{\#\,\mathrm{Prediction} = \mathrm{ActualLabel}}{\#\,\mathrm{TotalData}}$$

and the prediction is given by the corresponding category of the index of maximal model output. To give an overall evaluation of the model, the project concentrates both on reducing the training loss as well as improving the accuracy respectively on validation set and test set. Also, to visualize the final report, the team decides to plot the normalized confusion matrix for the classification results on the test data. All these metrics would contribute to the evaluation of the model.

3.3 Optimization

3.3.1 Optimizer Selection

In the project, the team has primarily utilizes the optimizers of Adams, Stochastic Gradient Descent (with momentum), Adagrad and RMSProp. After 25 epochs, each of the optimizers gives excellent models with similar accuracies (around 97.5% on both validation and test set). For epoch numbers equal to $1, 5, 10, 15, 20, 25$, we compare the results of each optimizer ($\mathrm{epoch} = 1$ shows the initial case):
The teams discover that the convergence speed is quite fast for all of the optimizers. In the implementation of the project, the team uses *torch.optim* to utilize the optimizers. The fast convergence can also be found in (figure 5) the plot of model losses at different checkpoints for all the optimizer candidates. In the following experiments, the team chooses Adam algorithm with learning rate 0.002 for further analyses.

[3]Thanks to the website: http://alexlenail.me/NN-SVG/LeNet.html
[4]For further information, check https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html

Table 1: Comparisons of Performance

Validation Accuracy % / Epochs / Optimizer	1	5	10	15	20	25	Test Accuracy(%)
Adam with learning rate 0.002	20.4	94.4	95.6	96.5	97.1	97.4	97.6
SGD with learning rate 0.1, momentum 0.9	10.5	94.2	95.7	96.4	96.7	97.1	97.4
Adagrad with learning rate 0.01	17.4	94.0	95.6	96.4	97.0	97.1	97.4
RMSProp with learning rate 0.001, $\alpha = 0.99$	17.3	94.4	95.8	96.6	97.0	97.3	97.4

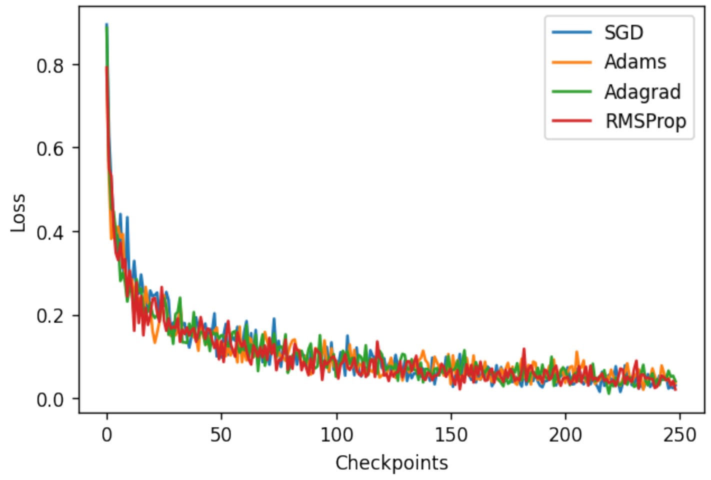


Figure 5: Loss versus Checkpoints for 4 Optimizers

3.3.2 Number of Epochs

Upon the choice of epoch numbers, the team first tries epochs of 50. After observing the changes in training and validation accuracy, the team discovers that the improvement in the model accuracy becomes insignificant and the trained model is susceptible to over-fitting. Based on an early stopping principle, the team decides to decrease the training epochs to 25.

3.3.3 Other Optimization Methods

Apart from multiple selections of optimizers, the team also implements other methods to achieve a more efficient training of the constructed CNN. To accelerate the training process, the team applied Batch Normalization in each convolutional layer (2d Batch Normalization) and between fully connected layers (1d Batch Normalization). Moreover, in the initial trials, the team faced the challenges from over-fitting: the model could only recognize pictures from the dataset. While predicting, the model will perform a random guess to the newly created data which has never appeared in the existing dataset. The team then applies the technique of *Dropout*, adding drop-out mechanisms to each of the fully connected layers, equipping the model with better generalization performance.

4 Experimental Results

The performance of the trained CNN is quite impressive. In the final rounds of experiments, the model has accuracies of around $97.3\% \sim 98.1\%$ on the test data (the randomness is given by the dropping-out mechanism and different optimization algorithms). This section gives a presentation of the experimental results from a data-based (internal) view and an external view (with extra data).

4.1 Data-Based Performance

Firstly, the plot of the accuracy curve with respect to the training epochs and checkpoints (see figure 6 and figure 7) presents a drastic speed of convergence and a stably growing trend of accuracy. We take 10 checkpoints within each epoch, and the two figures each respectively portrays the accuracy tendency from $1 \sim 25$ epoch and $2 \sim 25$ epoch due to the fast convergence in initial stages of the training. The fluctuation (of the training accuracy) is a result of training using batches of data (in this experiment, the team chooses the batch size to be 512). Additionally, the team discovers that the climax of the training accuracy (on the batches) reaches 1 for all the optimizers.

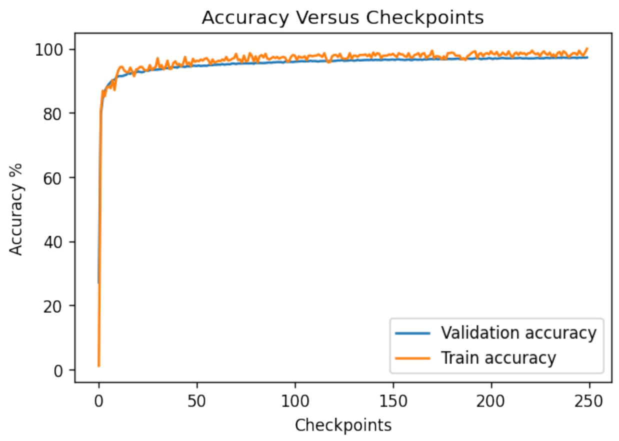


Figure 6: Accuracy w.r.t. Checkpoints

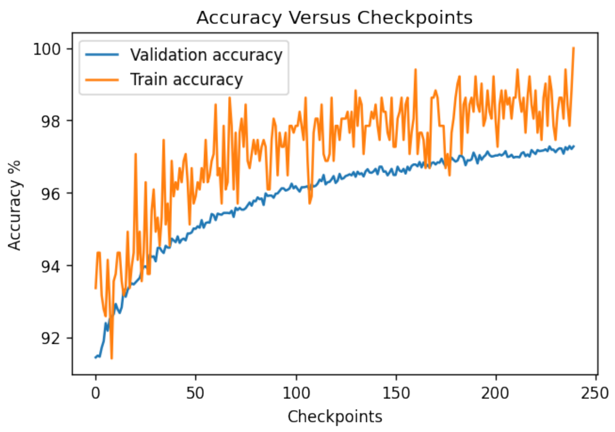


Figure 7: Accuracy w.r.t. Checkpoints [Truncated, starting at Epoch 2]

Apart from the numeric metrics, the confusion matrix generated by results on the test data is presented in figure 8. The confusion matrix gives a more direct visualization of the model accuracy. It's persuasive from the figure that the CNN model is successful in correctly classification: the diagonal is mostly colored by the darkest of the color map.

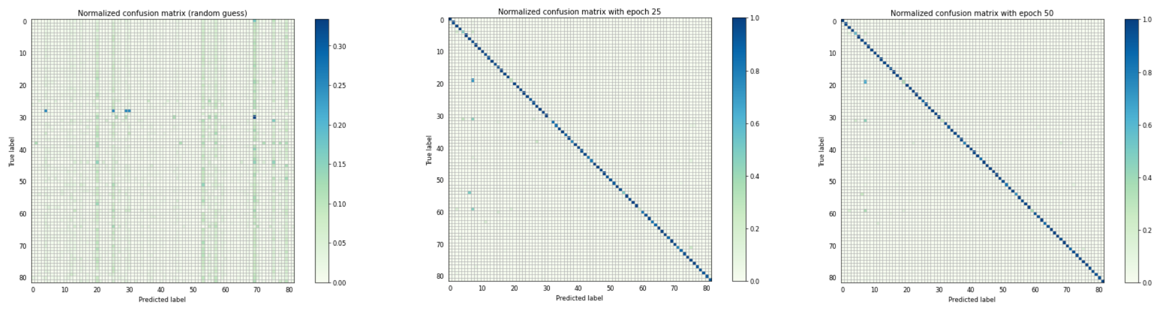


Figure 8: Comparison of Confusion Matrix of Test Data Predictions (Untrained, 25-epoch, 50-epoch)

In figure 8, the first matrix is the random guess by the model without training, and the performance improves significantly after 25 training epochs, as shown by the figure in the middle. The third matrix is the confusion matrix of model prediction after 50 epochs of training. Compared with the second picture, the model does not witness an impressive progress in the prediction results after extra 25 epochs of training. Hence, the choice of 25 epochs not only generates a robust model, but also avoids unnecessary computational cost.

4.2 Extra Data Performance

To test the model's robustness, the team designs some own hand-written mathematical symbols to simulate the practical application of the symbol. Figure 9 presents some successful identification of the mathematical symbols given by the model. The team creates new hand-written figures like $\int, \infty, 4, \theta, \pm$ in different styles from the dataset one. In such examples, the model successfully classifies the new inputs, showing its robustness in not being constrained by the training datasets.

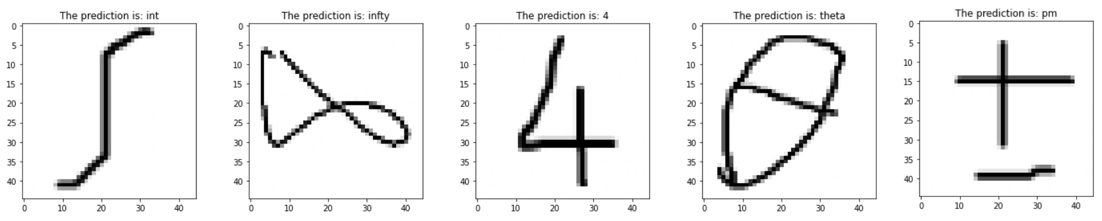


Figure 9: Model Predictions of New Data

5 Limitations, Future Works and Conclusions

5.1 Limitations

In terms of the deficiency of the model, one essential aspect is that the model is constrained by the comparatively monotone format of the chosen dataset. The pictures in the dataset the team analyzes share similar features: located at the center of the picture, no interference of other figures; also, the figures do not contain portraits with various thickness. The dataset is idealistic since it can't cover all the circumstances the model would encounter in practical applications. For instance, if the figure is wide, or it is hard to center the symbol in a square, or if two symbols are too close to be separated, this model may fail to give a good prediction.

Additionally, the training dataset, although has a considerable reservoir of samples, has an imbalanced distribution of symbols. For instance, the sizes of samples in numerical numbers (i.e. $0, 1, \cdots, 9$) reach over $5,000$ while categories like $\exists$ only have less than 100 samples. It results the model's failure to train on and identify symbols with small sample sizes. Moreover, the accuracy metric could only account for overall performance, making it more likely to neglect the unsatisfying performance on those classes in minority.

5.2 Future Work

Regarding the future work, the first part aims at solving the problems as described above. For example, the team will create more training data (via handwriting or Generative Adversarial Network) to the original one to resolve the imbalance. Also, a dataset with greater variety will be included for the further training and optimization. This part would perfect the existing experimental structure along with a better result.

Moreover, another aim is to enhance the model with more practical meanings. For instance, though the model is currently good at single-object detection and classification, the team will extend its impressive performance to multiple object detection (e.g. given a whole page with math symbols, the model would be able to return a well recognition of the overall page). A demo of our future work results is presented in figure 10.

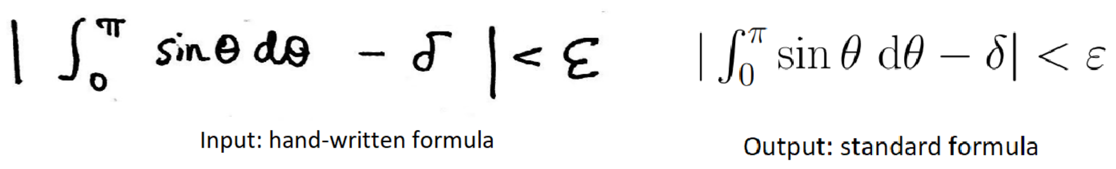

$| \int_0^\pi \sin\theta \, d\theta - \delta | < \varepsilon$
Input: hand-written formula
Output: standard formula

Figure 10: Demo: Identifying more Complex Formulas

5.3 Conclusion

The experiment demonstrates the outstanding performance of Convolutional Neural Network structures in analyzing pictures and carrying out different tasks. Also, the team has discovered that: deep learning projects are likely to be constrained by a specific training dataset, guiding the team to put future endeavours into diversifying the data to implement more robust models. Standing on giants' shoulders is how we can see further; the team would continually explore into essentials of machine learning and deep learning to obtain a more thorough understanding of them. The team would try to contribute to the advancement of their applications, expecting to make practical contributions in the future.

Acknowledgements

The team would like to express our great appreciation to Professor Guo and Professor Tam for their well-prepared, inspiring lectures as well as their valuable, constructive suggestions during the planning and development of this research work.

References

[1] Yann LeCun, Leon Bottou, Yoshua Bengio and Patrick Haffner. *Gradient-Based Learning Applied to Document Recognition*. IEEE, 1998.

[2] Xai Nano (at Kaggle). *Handwritten math symbols dataset*.
Link: `https://www.kaggle.com/xainano/handwrittenmathsymbols`.

[3] Alex Nail. Publication-ready NN-architecture schematics.
Link: `http://alexlenail.me/NN-SVG/index.html`.

[4] Pytorch Official Documents. *CROSS ENTROPY LOSS*.
Link: `https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html`.