Classification of Mathematical Symbols

Joshua Cappelli, Isabel N. Rivera Santiago

***Abstract*—Realizing the difficulty in deciphering handwritten mathematical symbols due to the variety of individual handwriting and the complexity of mathematical writing, this study explores the strengths and weaknesses of different preprocessing steps and convolutional neural networks in classifying different sets of handwritten math notation. We employ a transfer learning approach, leveraging both Xception and MobileNetV2 pre-trained on ImageNet. The model is fine-tuned on a dataset that undergoes preprocessing steps such as normalization and several augmentations to enhance the dataset and minimize overfitting. Our results show a significant improvement in classification accuracy compared to other machine learning techniques, achieving an accuracy of 88% on a small test set. The study confirms the potential of convolutional networks on mathematical notation recognition.**

***Index Terms*—classification, CNN, SVM, transfer learning**

I. Introduction

The challenge of interpreting different types of handwritten mathematical notation can be difficult due to the various factors that can play a role in affecting the strength of a model. This study focuses on exploring the efficacy of different preprocessing methods and convolution neural network (CNN) architecture to classify the different sets of handwritten mathematical symbols. Using various transfer learning pre-trained models, namely MobileNetV2, and Xception along with fine-tuning different preprocessing aspects we aim to enhance the ability to classify and recognize mathematical notation accurately [2][5]. Preprocessing steps include normalization and augmentation techniques such as rotations and reflections to refine the dataset and mitigate overfitting while the normalization aims to standardize the data. The primary study of this project is to investigate the effects of different preprocessing strategies and CNN architectures in the context of handwritten mathematical notation recognition and to evaluate their performance. In this paper, we present the results of the various experiments, along with demonstrating the capabilities of our models on the recognition of mathematical symbols.

II. Implementation

A. Preparation

We begin by collecting an assortment of handwritten mathematical symbols, namely [the variable *x*, square root, plus sign, negative sign, equal, percent, partial, product, pi, and summation], and then each symbol is labeled with its corresponding name.

B. Preprocessing

Before we feed the data into the CNN models, there are several preprocessing steps we attempt to try and enhance the quality of uniformity of the dataset to make a more robust model. This included normalization by standardizing each pixel from 0 to 1 to mitigate variations in the brightness and contrast of each image. Additionally, we did several data augmentation techniques such as rotations by 90-degree increments and inverting random images. This helped increase the diversity of our dataset and improve the model against different variations in handwriting styles. Finally, we did a 20 percent split for both our validation dataset and our test dataset without augmentation to prevent data leakage.

C. Model Architecture

We employ transfer learning to leverage the pre-trained convolutional neural network models on the ImageNet dataset [7]. We specifically utilized Xception and MobileNetV2 due to their efficiency, strength, and effectiveness in completing the task. In our final model, we used Xception as it returned the highest accuracy. For the base model, we used the weights for ImageNet with an input shape of (100,100,3) and then we froze the base model. In our layers, we used global average pooling, a dropout rate of 0.5, and the output layer was a dense layer with a softmax activation function. We also used the Adam optimizer along with the sparse categorical cross entropy (CCE) for our loss function.

The Adam optimizer is short for adaptive moment estimation [6]. It is an optimization algorithm that is mainly used for deep neural networks. It combines other averages for each parameter. Namely, the first moment of the gradients which corresponds to the mean, and the second moment of the gradients which corresponds to the variance. These two are used to compute the learning rates for the parameters. Since they are adaptive the learning rates can be adjusted according to the behavior of the gradients. In addition, Sparse CCE is a loss function that attempts to minimize the difference between the predicted probability distribution and the true probability distribution of the classes. For tasks such as this study, which is a multi-class classification, this type of loss function is often used. It calculates the cross-entropy between the true label and the predicted probabilities where the labels represent the classes which in our case are the different mathematical symbols.

D. Hyperparameter Tuning

We used several different strategies to try and optimize the performance of our model. Some examples were using 32 and 64 as batch sizes along with different amounts of epochs to ensure the optimal configuration of our model for the dataset. Finally, we tested using different activation functions such as ‘relu’ and ‘selu’. We also used different learning rates for each of our models to mitigate overfitting and find the best weights for our models.

TABLE I
OVERVIEW OF EXPERIMENTS.

Model	Test Accuracy	Optimizer	Loss function
Base CNN	67 %	Adam	SCCE
SVM + CNN	61 %		CCE
MobileNetV2	85 %	Adam	CCE
Xception	88 %	Adam	SCCE

III. EXPERIMENTS

Our first convolution neural network model, in which we only reformatted each image to 72,72 to remove some unnecessary features as well as reduce the computational complexity while preserving information regarding each image. We also normalized each image so that each grayscale pixel value was between 0 and 1 which lowers the contrast in the image. Furthermore, the neural network was trained using 4 layers in the first had 64 filters and the next had 128 filters. We also had a scheduler to lower the learning rate over time to mitigate overfitting and used the sparse categorical cross entropy for the loss function.

In our second model which was also a CNN, we attempted to find the misclassified data using an support vector machine (SVM)[1]. After normalization, we ran the SVM model and deleted any sample that was misclassified. We then ran the same steps as we did in the first model to build our CNN.

In our third model, we used transfer learning to train our CNN, implementing MobileNet[2]. MobileNet is a CNN architecture that is specifically made for deployment on devices that have limited computational power. The key difference is its use of depthwise separable convolutions which reduce the amount of parameters and thus the computational cost compared to other convolution layers. We first downscaled to 96 by 96 and then for preprocessing we both normalized and rotated randomly many of the images to create new images. We tried several different strategies to improve the test accuracy such as implementing a learning rate scheduler and using categorical cross entropy rather than sparse categorical cross entropy [3][4]. The difference between the two is that with sparse, each integer label is seen as a separate class while with normal categorical cross entropy is represented by one hot encoded vector [9].

In our final model, we used transfer learning from ImageNet to train our CNN, more specifically we implemented Xception. Xception stands for extreme inception and it is a deep CNN architecture that is specifically effective at feature extraction tasks such as our image classification task. The difference with Xception is its use of both depthwise convolution and pointwise convolution also known as 1x1 convolution which projects depthwise features into a new feature space. Using the same preprocessing techniques as our previous model without downscaling any image. We also regularized with a dropout of 0.5 and had a learning rate of 0.0001, a beta of 0.9, and a beta 2 of 0.999. All test set accuracy values are listed below in Table 1.

width=0.52024-04-24 235207.jpg

IV. CONCLUSION

In this study, we investigated how different methods of preprocessing and models can be used in the recognition of handwritten mathematical symbols. We also addressed the different challenges, strengths, and shortcomings of different structures and how they can be used to better train a model for recognition tasks. By employing and leveraging transfer learning and different preprocessing techniques and data augmentation we achieved notable results compared to simpler or more traditional methods in machine learning. Our experiments demonstrate the effectiveness of CNNs on classification problems with an accuracy of 88 percent on our test set. This shows the potential of deep learning techniques for classification tasks.

Furthermore, our analysis of the different preprocessing strategies gives strong insight into the influence they hold over the strength of the model against various handwriting styles and images. While our experiments hold strong results one area of exploration for future results and improvements for this task would be to employ CleanLab.io to better filter the training data for finding and fixing of mislabeled or misclassified data[8]. This would help strengthen the power of the model as misclassified data can seriously hinder the capabilities of a model. In conclusion, our results reaffirm the strength of convolutional neural networks in addressing classification tasks such as handwritten mathematical symbol recognition. By combining different machine learning concepts such as transfer learning and CNN we have made significant and reliable results.

ACKNOWLEDGMENT

The authors would like to thank Dr. Catia Silva and the Teaching Assistants.

REFERENCES

[1] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt and B. Scholkopf, "Support vector machines," in IEEE Intelligent Systems and their Applications, vol. 13, no. 4, pp. 18-28, July-Aug. 1998, doi: 10.1109/5254.708428.

[2] A. G. Howard *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv.org*, 2017. https://arxiv.org/abs/1704.04861https://arxiv.org/abs/1704.04861

[3] A. Mao, M. Mohri, and Y. Zhong, "Cross-Entropy Loss Functions: Theoretical Analysis and Applications," *arXiv.org*, Apr. 14, 2023. https://arxiv.org/abs/2304.07288https://arxiv.org/abs/2304.07288

[4] TensorFlow, "tf.keras.losses.SparseCategoricalCrossentropy — TensorFlow Core r2.0," *TensorFlow*, 2019. https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy

[5] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," *arXiv.org*, 2016. https://arxiv.org/abs/1610.02357https://arxiv.org/abs/1610.02357

[6] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv.org*, Dec. 22, 2014. https://arxiv.org/abs/1412.6980https://arxiv.org/abs/1412.6980

[7] O. Russakovsky *et al.*, "ImageNet Large Scale Visual Recognition Challenge," *arXiv.org*, 2014. https://arxiv.org/abs/1409.0575https://arxiv.org/abs/1409.0575

[8] "Cleanlab Open-Source Documentation," *cleanlab*. https://docs.cleanlab.ai/stable/index.html (accessed Apr. 25, 2024).

[9] Z. Zhang and M. R. Sabuncu, "Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels," *arXiv:1805.07836 [cs, stat]*, Nov. 2018, Available: https://arxiv.org/abs/1805.07836