Handwritten Symbol Classification using Convolutional Neural Network

Mohit Palliyil Sathyaseelan[1] , Priyanka Abhijit Tamhankar[1]
[1] Electrical and Computer Engineering School , University of Florida.

Abstract—**The report explains the process of designing a Convolutional Neural Network for Handwritten symbol classification. It explains the process that went under training a Machine Learning Model to accurately classify handwritten symbols into 25 classes. Upon following the experimental design approach and testing on different models it was concluded that convolutional neural networks gave the best possible accuracy in classifying symbols from the provided dataset.**

Keywords— Convolutional Neural Networks, Machine Learning.

I. Introduction

The project deals with classifying a custom-made data set of approximately twenty-three thousand five hundred and two images into twenty-five classes. Our objective for the project was to predict which class the symbols belong to amongst the twenty-five. Being a data set that contained certain noise, extracting necessary features, processing the data to eliminate noise, selecting an appropriate model, and designing a neural network to train the model to correctly identify symbols were the major tasks. After going through various research papers and articles, and experimenting on those methods, CNN was selected to train our model for this application. Comparing KNN and CNN we found that CNN provided a better accuracy [1]. Also [2] compares various other algorithmic models such as SVM, Back Propagation, etc. Finally, CNN provided the best accuracy for the provided data set.

II. Implementation

The combined data set had 23,502 images, so according to our experimental design we split the training and testing data into 9:1 ratio. Since we were using Pytorch, it was convenient to handle data using dataLoader object. The CNN we designed takes a 4d array as an input, therefore we reshaped the data accordingly. Our next goal was to implement image processing techniques mentioned in the experimental design, (i.e.) Gaussian blur and Interpolation. For gaussian blur, the kernel size was the parameter. We set this to 7 to prevent diminution of Image data. With respect to interpolation, we used the library function that uses nearest neighbors algorithm.

In the next part of implementation, we passed the data of size 28x28 after image processing to the Convolutional Neural Network. The CNN model we are using has nine Convolutional layers, 6 normalization layers, 2 pooling layers and finally an average pooling layer. Except from the last convolution layer, they all use the ReLu activation function. The last convolution layer uses SoftMax activation function. The Network was designed from experimental research and with the help of pytorch.nn.function documentation[3]. The layer outputs were calculated with respect to their function formulas. (See Fig1.).

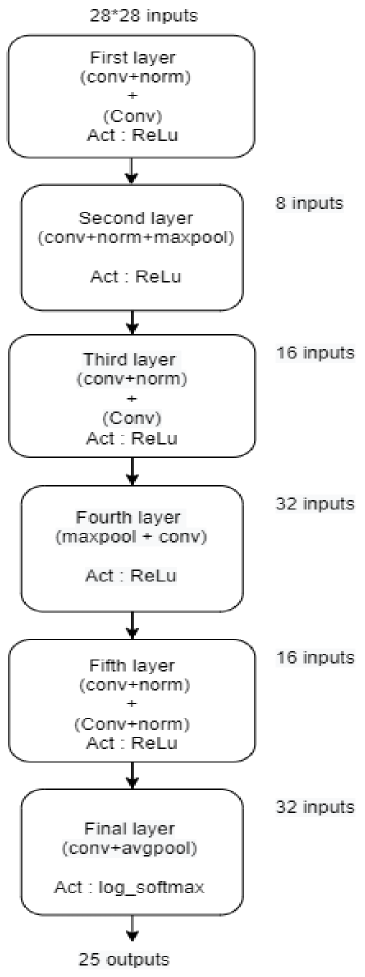
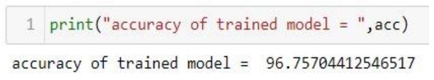

28*28 inputs
First layer (conv+norm) + (Conv) Act : ReLu
Second layer (conv+norm+maxpool) Act : ReLu
8 inputs
Third layer (conv+norm) + (Conv) Act : ReLu
16 inputs
Fourth layer (maxpool + conv) Act : ReLu
32 inputs
Fifth layer (conv+norm) + (Conv+norm) Act : ReLu
16 inputs
Final layer (conv+avgpool) Act : log_softmax
32 inputs
25 outputs

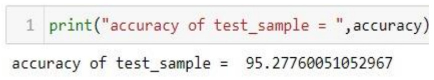
Fig1. CNN structure and layers

The output from the model gave us the predicted labels which ranges between [0,24]. The model performance was evaluated with the validation data set in the training phase which resulted to be 96.7% accurate. A threshold was set for the model as mentioned in the experimental design. The model was saved after training phase. In the testing phase, the pre-trained model was loaded, and accuracy was evaluated on test data which came out to be 95.27%



```
print("accuracy of trained model = ",acc)
accuracy of trained model =  96.75704412546517
```

Fig2. Accuracy of trained model



```
print("accuracy of test_sample = ",accuracy)
accuracy of test_sample =  95.27760051052967
```

Fig3. Accuracy of test data

III. Experiments

Initially, we started training our model with K-nearest neighbors but it resulted in low accuracy. We tried using different combinations of hyperparameters like number of neighbours and varying range of train and test data sets. We used Histogram of Gradients (HOG), in order to get better-processed images for the KNN model. Finally with changing all possible parameters for the model we got an accuracy of 56% on the trained data set.

On further investigation, we concurred that a Neural Network provides better accuracy than the KNN model. Our research dealt with the handwritten digits recognition projects on the MNIST data set[4]. We designed a simple Linear Neural Network with two hidden layers as a starting point and checked the accuracy. On fine-tuning the model with the parameters, we got decent accuracy. A Neural Network works better on large training data, hence this was a good approach for us. We further investigated different filters in Pytorch that could be used on our model. Although this did not give us the accuracy we were targeting.

The next experiment was to use convolutional layers in our Neural network. Convolutional Neural Networks provide better feature extraction than that of a simple Neural Network. It also has advantages like pre-training and weight sharing. Designing a good Convolution network was our next goal. The images were pixelated to 150x150 and we had to classify them into 25 classes, hence the CNN took in 22500 features as input and later returns an array of the labels or classes. Our feature extraction layers in CNN had 4 convolution layers with pooling layers, respectively. As we expected our model did improve to give a better accuracy when compared to a simple linear NN (Neural Network). After observing results on the experiments done on MNIST data set [5], when we compared a CNN to a simple Neural network, the accuracy improves by a considerable margin. This was not the case with our model, hence we pre-processed the images, before passing it through the Neural Network. One of the main reasons why we converted our image data into 28x28 was the image property. If our image data had detailed features likes feathers, eyes etc, this model would not work accurately. Therefore, we tried running various image processing techniques to extract features from the data. We had to be sure that the images did not lose the prominent features that define the structure of individual symbols which differentiate them from each other. [6] We used Interpolation transformation to preprocess the data. Given below are the results we obtained when we processed the images using the interpolation function (See Fig4). We used non-adaptive interpolation which treats all pixels equally and uses k nearest neighbors to resize the image without losing its identity or cropping image data.

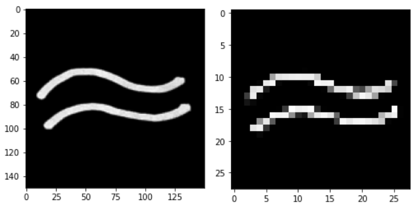


Fig4. Interpolated Image

But, when we further tested all the symbols with the interpolation, we observed that it was losing important data and misclassifying symbols that had similar horizontal or vertical orientations. Like for example, in the figure below Fig.5(b), after interpolation, might be classified as a not-equal-to sign ($\neq$). Therefore, we needed a better image pre-processing technique. On further research in the available pytorch documentation, we used the Gaussian filter as a preprocessing method [7]. It did a better job at retaining the features that defined uniqueness of every symbol. Using this filter, the model accuracy on train data boosted by 7%, increasing it to 96.7%. We set a threshold on the number of epochs in order to prevent overfitting. Further, on testing the pre-trained model with test data, we obtained an accuracy of 95%.

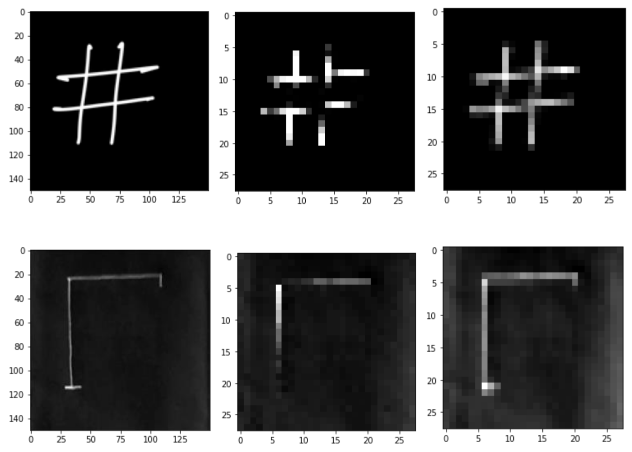


Fig5. (a) True image (b) Interpolated output image (c) Gaussian and Interpolated output Image

The final experiment was to decide how to split the dataset into train and test. First and foremost, the data was shuffled, and appropriate targets were mapped to images for training. The size of the training set is a parameter of both the data and the structure and complexity of the model. Hence, iterating over different combinations and testing the accuracy for each was the best way to come up with the size of training data.

Sr. No.	Training data (%)	Accuracy of the model
1	90%	96.1%
2	80%	92.8%
3	70%	92.5%

Fig6. Accuracy of model with different splitting ratio of data set

Hence, running the model on different ratios of data set and analyzing the changes in accuracy on test data, the test data was divided as 10% of the total data set and 90% was used for training.

IV. Conclusion

From the above experiments, it can be concluded that constructing a clean data set that enhances features useful for differentiating between classes is the elementary step one should follow while designing a good classification model. One important thing that we learnt was there is no such thing as a perfect model for a particular problem. Designing a good model comes down to how well we tackle the challenge of extracting essential parameters from data and rigorous experimentation. We also learnt that Convolutional Neural Network performs better with a bigger data size and one of the biggest advantages of using CNN is the flexibility of adding filters to the layers. The scope of customization and possible models in CNN is what makes it the good fit for image classification. By refining the filters with their parameters, we achieved 95% accuracy for our model.

References

[1] T. Makkar, Y. Kumar, A. K. Dubey, Á. Rocha and A. Goyal, "Analogizing time complexity of KNN and CNN in recognizing handwritten digits," 2017 Fourth International Conference on Image Information Processing (ICIIP), 2017, pp. 1-6, doi: 10.1109/ICIIP.2017.8313707.

[2] W. Liu, J. Wei and Q. Meng, "Comparisions on KNN, SVM, BP and the CNN for Handwritten Digit Recognition," 2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications(AEECA), 2020, pp. 587-590, doi: 10.1109/AEECA49918.2020.9213482.

[3] Pytorch.org. *TORCH.NN.FUNCTIONAL*. Available: https://pytorch.org/docs/stable/nn.functional.html

[4] Amitrajit Bose, "Handwritten Digit Recognition Using PyTorch — Intro To Neural Networks"

[5] Ravi Vaishnav, "Handwritten Digit Recognition Using PyTorch"

[6] O. Rukundo and B. T. Maharaj, "Optimization of image interpolation based on nearest neighbour algorithm," 2014 International Conference on Computer Vision Theory and Applications (VISAPP), 2014, pp. 641-647. .

[7] pytorch.org.. *GAUSSIANBLUR*. Available: https://pytorch.org/vision/master/generated/torchvision.transforms.GaussianBlur.html