Classification of Handwritten Symbols Using Convolutional Neural Networks

Kunal Dudhe[1], Aakash Naik[1], and Anurag Patil[1]

[1]University of Florida

January 11, 2022

Abstract

Image classification is one of the oldest machine learning problems that fall under the category of supervised learning. Over the years, significant progress has been made in developing algorithms that can attain high accuracy while classifying an image. Convolutional Neural Networks are widely used for image classification problems in machine learning. In this work, we have used a CNN in order to classify a dataset consisting of 23502 handwritten mathematical symbols. We found that CNN along with NAdam optimizer produced fairly accurate results.

Keywords: Convolutional Neural Networks (CNN), kNN, epochs, conv2D layer, ReLU, max pool layer, fully connected layer, optimizers, filters

Introduction

Convolutional Neural Networks are known to achieve impressive performance in visual recognition problems [1]. This report elaborates on our goal of implementing a classifier that is used to classify handwritten symbols. We present our findings after using a Convolutional Neural Network (CNN) as our classifier for the same purpose.

Handwritten symbol classification is a well-researched area and there are various suggested methods to perform the classification. SVM with radial basis kernel function (LIBSVM) yields a 79.19% accuracy, while a system using Support Vector Machine has obtained an accuracy of 83.61% [1]. Bi-directional long short term memory recurrent neural network has achieved best results with 84.14% accuracy [1], while kNN classifier using HOG descriptor for word recognition system has achieved an accuracy of 80.01% [2].

This report includes the results of a comparison between the performance of K Nearest Neighbors and Convolutional Neural Networks for handwritten symbol classification. We trained several different architectures of CNN to get better classification accuracy. The training of the developed CNN model was performed using a custom dataset of 23502 images and classified into 25 distinct classes. After comparing the accuracy scores, we concluded that the accuracy scores obtained using CNN were significantly better than those obtained using the KNN algorithm.

Implementation

CNN is a type of multilayer perceptron and combines three architectural ideas to ensure some degree of shift, scale, and distortion invariance: local receptive fields, shared weights, and spatial subsampling[3]. The CNN architecture constructed for the experiment consists of a combination of Convolution- ReLU(Activation function) - Pooling layers, followed by a Fully connected layer.

Convolution is an image processing technique used to perform edge detection to identify pixels with discontinuities in an image.

Rectified linear activation function or ReLU is used to increase the non-linearity in our convolution output. Non-linearity is desired because we need a non-linear decision boundary to classify the symbols.

Pooling is used to reduce the size of an output image by combining the output of pixels in a pooling window to a single output pixel.

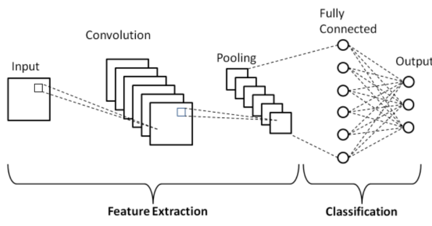


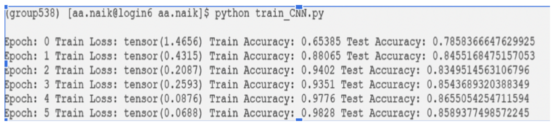
Figure 1: CNN overview

This can be done by taking an average of values in that window (average pooling) or keeping only the maximum value (max pooling). We have used max-pooling for our experiment because it helps extract low-level features like edges, points, etc.

Optimizers are used while training a neural network to reduce the loss. Pytorch lets us use various optimizers, but adaptive-based optimizers achieved the highest results for image classification problems [4]. We have used NAdam (Nesterov - Accelerated Adaptive Moment Estimation) optimizer in our implementation.

The designed CNN model has the following architecture: Convolution layer: We used four convolution layers in our architecture. Each of these layers has a filter/kernel size of 3. Each of these layers has two parameters called in_channels (IC) and out_channels (OC).

- **First convolution layer**: IC=1 and OC=15.
- **Second convolution layer**: IC=15 and OC=30.
- **Third convolution layer**: IC=30 and OC=45.
- **Fourth convolution layer**: IC=45 and OC=60.
- **ReLU layer**: We have used a ReLU layer after each convolution layer.
- **Max pooling layer**: A single max-pooling layer is used with a kernel size of 3 to reduce the image size from 150 * 150 to 50 * 50
- **Fully connected layer**: After the final ReLU layer, we flatten the image into a 1D array to be fed to the fully connected layer. This layer has in_featuress=50 * 50 * 60 and out_channels=25. The in_features are decided by the max-pooling layer and the number of channels in the convolution layers while the out_channels are equal to the number of classes.



```
(group538) [aa.naik@login6 aa.naik]$ python train_CNN.py

Epoch: 0 Train Loss: tensor(1.4656) Train Accuracy: 0.65385 Test Accuracy: 0.7858366647629925
Epoch: 1 Train Loss: tensor(0.4315) Train Accuracy: 0.88065 Test Accuracy: 0.8455168475157053
Epoch: 2 Train Loss: tensor(0.2087) Train Accuracy: 0.9402 Test Accuracy: 0.8349514563106796
Epoch: 3 Train Loss: tensor(0.2593) Train Accuracy: 0.9351 Test Accuracy: 0.8543689320388349
Epoch: 4 Train Loss: tensor(0.0876) Train Accuracy: 0.9776 Test Accuracy: 0.8655054254711594
Epoch: 5 Train Loss: tensor(0.0688) Train Accuracy: 0.9828 Test Accuracy: 0.8589377498572245
```

Figure 2: Training & test accuracy per epoch

Experimental Design

The experiment was performed using a dataset consisting of 23,502 handwritten mathematical symbols. In order to select the best algorithm for image classification, various combinations of hyperparameters therein were implemented. The subsequent results obtained aided us in finalizing the algorithm and the hyperparameter values within them. In order to compare the performance, we used k-NN initially and then built a CNN model using the PyTorch library.

Detailed explanation:

Train-Validation-Test split:
The data was split into three parts: Training set, validation set, and testing set, comprising 80%, 10% and 10% of the dataset, respectively. The validation dataset was used for the purpose of tuning the hyperparameters.

Implementation of KNN:
The K-Nearest Neighbor classifier with the HOG descriptor was implemented in order to classify images. The k-NN algorithm was implemented with the following parameters:

- K = 1
- Orientations = 16
- Pixels per cell = 8*8
- Cells per block = 5*5
- A total of 78400 features were extracted per image.

For one of the runs on the test data, an accuracy of 73.42% was observed.
For multiple runs of the algorithm, the accuracy score ranged in between 71% to 75%.

Implemention of CNN:
The convolutional neural network contains the following layers in the order specified below:

- Conv2D Layer $\longrightarrow$ Batch Normalization $\longrightarrow$ ReLU Layer
- MaxPool Layer
- Conv2D Layer $\longrightarrow$ ReLU Layer

- Conv2D Layer $\longrightarrow$ Batch Normalization $\longrightarrow$ ReLU Layer
- Conv2D Layer $\longrightarrow$ ReLU Layer
- Fully Connected Layer

The hyperparameters in consideration were:

- Number of Layers
- Number of Neurons
- Batch Size
- Epoch
- Activation Function
- Learning Rate
- Weight Decay
- Type of Loss Function
- Optimizer type

The number of layers, neurons, batch size, epochs were chosen randomly. The learning rate was set to 0.001, while weight decay was set to 0.0001.

The loss functions that were chosen for this experiment were Log Loss, Negative Log Loss, and Cross-Entropy Loss. We observed that the Cross-Entropy Loss produced the most desirable accuracy scores for multiple runs of the constructed CNN.

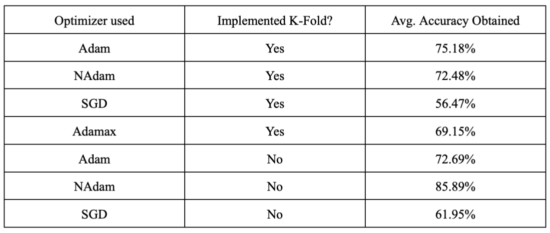
The optimizers that were used are SGD, Adam, Adamax, NAdam.

For further strengthening our experimental design, we also implemented the K-Fold Cross Validation strategy. While, generally, it is expected from a model to have a better accuracy score, the results of our experiment said otherwise.

As a result, a combination of various optimizers stated above, with/without K-Fold Cross Validation was used in the CNN algorithm.
The results are summarized in the table shown in Figure 3.

Softmax layer implemented with Adam and SGD optimizer after the Fully Connected Layer produced lower accuracy scores than expected. Hence, the softmax layer was dropped.

Finalizing the model: We found that our model produced the highest accuracy for the NAdam optimizer with no K-Fold cross-validation.



Optimizer used	Implemented K-Fold?	Avg. Accuracy Obtained
Adam	Yes	75.18%
NAdam	Yes	72.48%
SGD	Yes	56.47%
Adamax	Yes	69.15%
Adam	No	72.69%
NAdam	No	85.89%
SGD	No	61.95%

Figure 3: Experimental Results

Conclusions

In this work, we trained a Convolutional Neural Network model for mathematical symbol classification. For Arabic word recognition using k-NN with a HOG descriptor, the highest accuracy was observed when the number of neighbors k = 1 [2]. A similar result was observed in the experiments performed in this work. However, although k-NN was a good start, we observed that with CNN we could achieve higher accuracy through hyperparameter tuning.

The number of convolution layers, the type of activation functions used after each layer convolution layer, the presence/absence of batch normalization after the convolution layer along with the number of pooling layers affected the result. An increase in the number of convolution layers increased the accuracy of the model drastically. However, increasing the number of max pooling layers led to sub-par results on the test data. The number of neurons in the fully connected layer along with the number of layers themselves were another hyperparameter which needed extensive experimental tuning. In our work, we have settled on no hidden layers inside the fully connected layer. This was because adding one (or more) hidden layers led to a decrease in the performance of the model. Even though softmax layer is used as the output layer in CNN when performing classification, we observed that including it in our model decreased its accuracy drastically.

The type of optimizer often determined where our model would saturate. Using Stochastic Gradient Descent (SGD), resulted in poor performance in the initial few epochs until it saturated around 62%. Using Adam and NAdam increased the accuracy to 73% and 86% respectively. Learning rate and weight decay (L2 regularization), if in the order of 10^{-1} led to an early convergence, and often with an undesired accuracy. By using NAdam optimizer with learning_rate = 0.001 and weight_decay = 0.0001, we achieved a classification accuracy of 85.89%. We believe that our CNN model can be further fine tuned to achieve better classification results. The accuracy could be further increased by pre-processing the data using appropriate pre-processing methods.

References

[1] Irwansyah Ramadhan, Bedy Purnama, and Said Al Faraby. Convolutional neural networks applied to handwritten mathematical symbols classification. In *2016 4th International Conference on Information and Communication Technology (ICoICT)*, pages 1–4, 2016.

[2] Soufiane Hamida, Bouchaib Cherradi, and Hassan Ouajji. Handwritten arabic words recognition system based on hog and gabor filter descriptors. In *2020 1st International Conference on Innovative Research in Applied Science, Engineering and Technology (IRASET)*, pages 1–4, 2020.

[3] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[4] Ibrahem Kandel, Mauro Castelli, and Aleš Popovič. Comparative study of first order optimizers for image classification using convolutional neural networks on histopathology images. *Journal of Imaging*, 6(9), 2020.