Automatic Handwritten Digit Recognition for MNIST Dataset

1. PROBLEM STATEMENT AND RESEARCH QUESTION

Computer scientists recently have been working on automating several tasks, including tasks that were previously done by the human brain. Computation by Convolutional Neural Network (CNN) is inspired by the human brain. The human brain can perceive or identify objects visually. As humans, we train our children to recognize objects by showing them several pictures of that object. This helps the child's brain to identify or make a prediction about objects that he/she has never seen before. A CNN works in the same pattern and is popular for analyzing visual imagery. It adapts layered architecture where the layers represent mathematical functions, and each neutron is a carrier of weights. Layers that lie between input and output layers are called hidden layers, which can be changed based on the task. Convolution is a process of applying several filters on images to highlight the details in it. Each binary classifier called perceptron, many perceptron mappings from each neuron to every other neuron of forwarding layers together forms a neural network. A CNN fuses feature extraction phase and classification steps to successfully identify different categories of objects; in this case – handwritten digits from the MNIST dataset [1]. The MNIST dataset is an acronym that stands for the Modified National Institute of Standards and Technology dataset. It is a dataset of 60,000 small square 28×28-pixel grayscale images of handwritten single digits between 0 and 9. The task is to classify a given image of a handwritten digit into one of 10 classes representing integer values from 0 to 9, inclusively.

The research questions are:
Is CNN model efficient in classifying handwritten digits?
Does combining CNN and LSTM models significantly improve the performance of the CNN model?

2. LITERATURE SURVEY

Many techniques have been developed to recognize handwritten digits. Yue Yin, Wei Zhang [2] concluded that of all neural network implementations, the CNN method is valid for OCR based image classification systems. Mahmoud M. Abu Ghosh [3] compared CNN, DNN, DBN models to determine which neural network model is efficient in the field of computer vision, specifically in image classification like OCR. They claim that the DNN's (Deep Neural Network) accuracy beats other neural networks with 98.08% accuracy but falls behind CNN in terms of execution time.

In the past few years, the CNN model has been explored for handwritten digit recognition from the MNIST benchmark database. Some researchers have reported accuracy as good as 98% or 99% for handwritten digit recognition [4]. An ensemble model has been created using a combination of several CNN models. The recognition experiment was carried out for MNIST Sensors and an accuracy of 99.73% was reported [5]. An extraordinary recognition accuracy of 99.81% was reported by Niu and Suen by integrating the SVM (support vector machine) capability of minimizing the structural risk and the capability of a CNN model for extracting the deep features for the MNIST digit recognition experiment [6].

A.K. Jain [7] implemented an approximation based KNN classification algorithm for handwritted digit recognition. He matched two character's deforming edges and dissimilarities. The work proposed is on low-dimensional space patterns, where the scaling is 2000 times lesser results 99.25% accuracy.

R.Alhajj [8] introduced a novel approach called the agent-oriented approach. In summary, they appoint agents to each character in order to identify the hills and valleys, which are nothing but blacks and whites. The ability of agents to socialize with each other is highlighting features compared with any other image classification techniques. Overlapping digits are identified based on the cut-points, which is nothing but the intersection of agent paths. The results of their methods are surprisingly higher when compared to many other ANN-based approaches at about 97% accuracy.

In this project, I intend to compare the performance of two models CNN and CNN-LSTM with relu and sigmoid activation functions to predict a real-world handwritten digit - the MNIST dataset [1].

3. IMPLEMENTATION

The implementation phase can be divided into four stages. The first stage prepares the programming environment to begin the project. The second and third stage prepares the data set and creates the models respectively. The final stage trains and validates the models using the dataset provided in stage two. The stages are discussed in detail below:

a. Preparing the programming environment

In this stage, I provided a computer with a RAM of 8gb and installed PyCharm and python 3. Also, I installed the required modules; tensorflow, keras, numpy, and matplotlib for creating the models and for graph visualization.

b. Preparing the dataset

In this stage, I got the MNIST dataset from the python library. After loading the data, I separated the data into X and y where X is the image, and y is the label corresponding to X. Finally, I divided the dataset into training set and test set. An example of the MNIST dataset can be seen below:

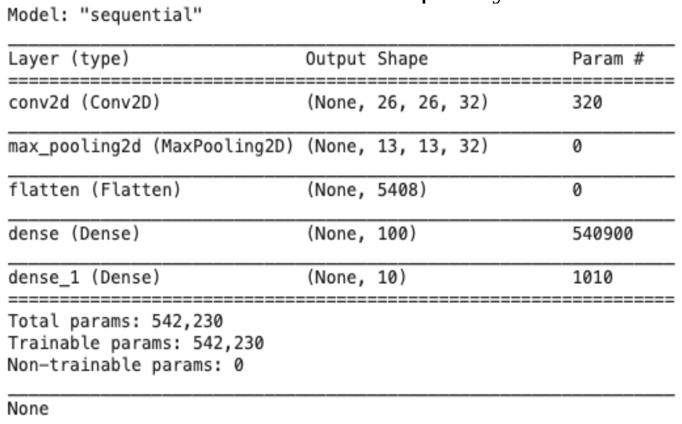
Figure 1: The MNIST dataset sample

c. Creating the Models

After the second stage, the images and labels are ready to be fitted into the models.

<u>Implementing the CNN model</u>

Convolution and max-pooling are done to extract the features in the image, and a 32 3x3 convolution filter is applied to a 28x28 image followed by a max-pooling layer of 2x2 pooling size followed by a flatten layer. Next, is a dense layer of 100 neurons, which is then connected to the softmax output layer of 10 neurons.



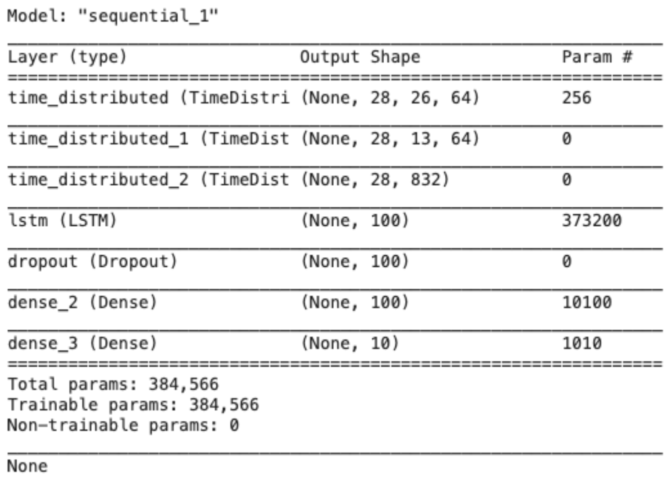
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 5408)              0
_________________________________________________________________
dense (Dense)                (None, 100)               540900
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1010
=================================================================
Total params: 542,230
Trainable params: 542,230
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 2a: CNN Model Summary

<u>Implementing the CNN-LSTM model</u>

For the CNN-LSTM model, I used the same CNN model except that the convolutional layer will be in 1-D for the shape to be compatible with the LSTM layer. Then I added the

LSTM model to it using keras with a dropout of 0.5 to reduce overfitting, just before the dense layer of 100 neurons, which is then connected to the softmax output layer of 10 neurons.



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
time_distributed (TimeDistri (None, 28, 26, 64)        256
_________________________________________________________________
time_distributed_1 (TimeDist (None, 28, 13, 64)        0
_________________________________________________________________
time_distributed_2 (TimeDist (None, 28, 832)           0
_________________________________________________________________
lstm (LSTM)                  (None, 100)               373200
_________________________________________________________________
dropout (Dropout)            (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 100)               10100
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1010
=================================================================
Total params: 384,566
Trainable params: 384,566
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 2b: CNN-LSTM Model Summary

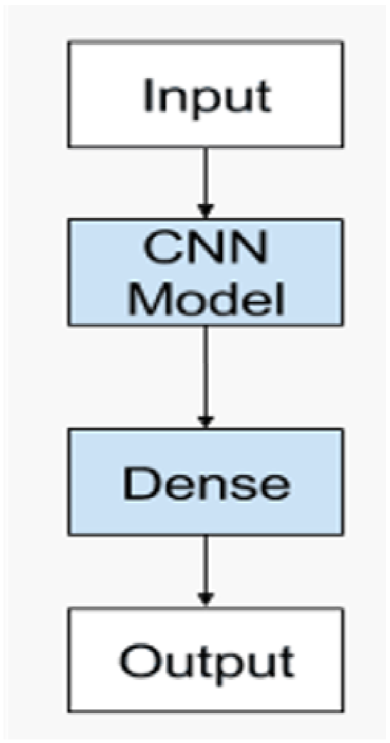


Figure 3a: CNN Architecture

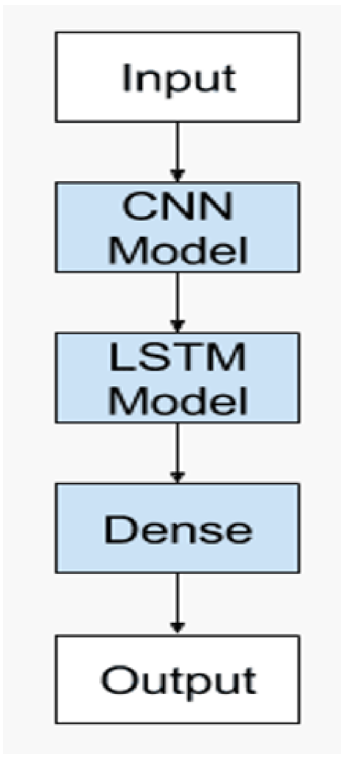


Figure 3b: CNN-LSTM Architecture

d. Training and Validation

After building the models, I compiled the models with Adam optimizer and cross-entropy loss function, which is standard in making a convolution neural network. Once the model was compiled, then I trained the model with training data for 12 epochs.

4. EXPERIMENTS

The losses and accuracies were measured for every epoch, for both the CNN model and CNN-LSTM model and the losses and accuracies were plotted and visualized in a graph for comparison.

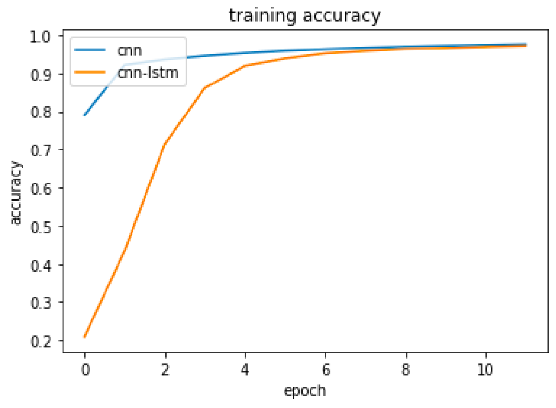


Figure 4a: Training Accuracy

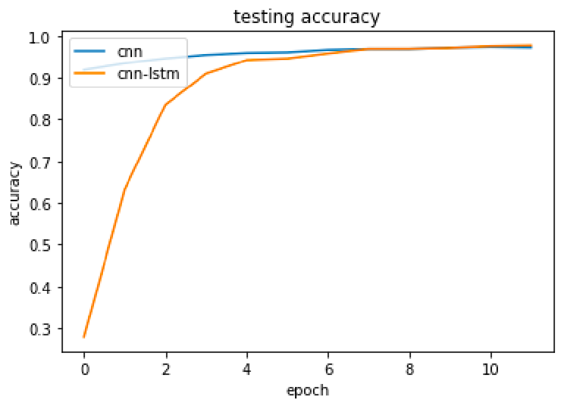


Figure 4b: Testing Accuracy

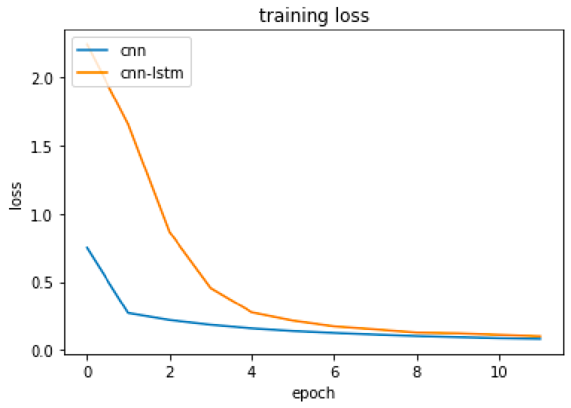


Figure 4c: Training Loss

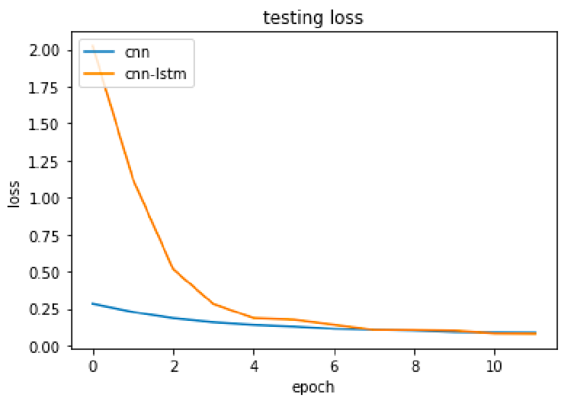


Figure 4d: Testing Loss

After training for 12 epochs, we can see in figures 4a-d above that the performance of the CNN model vs the CNN-LSTM model are quite similar. Also, both models recorded approximately 0.97 (**97%**) each in accuracy and 0.09 (9%) each in loss.

5. CONCLUSION

In summary, after creating two models: the CNN model and the CNN-LSTM model, and training each of them to classify 10 classes of handwritten digits from the MNIST dataset, I observed that both models performed similarly without much significant difference in performance.
Now, to answer the research questions mentioned in the introduction: The CNN model is effective for classifying the MNIST dataset. Also, adding the LSTM layer to create a CNN-LSTM model does not significantly improve the performance of the original CNN model.

6. REFERENCES

1. LeCun, Y.; Cortes, C.; Burges, C.J.C. The MNIST Database of Handwritten Digits. 2012. Available online: http://yann.lecun.com/exdb/mnist/ (accessed on 25 April 2018).
2. Y. Yin, W. Zhang, S. Hong, J. Yang, J. Xiong, and G. Gui, "Deep Learning-Aided OCR Techniques for Chinese Uppercase Characters in the Application of Internet of Things," in IEEE Access, vol. 7, pp. 47043-47049, 2019. https://doi.org/10.1109/ACCESS.2019.2909401
3. M. M. A. Ghosh and A. Y. Maghari, "A Comparative Study on Handwriting Digit Recognition Using Neural Networks," 2017 International Conference on Promising Electronic Technologies (ICPET), Deir El-Balah, 2017, pp. 77-81.
4. Decoste, D.; Schölkopf, B. Training invariant support vector machines. Mach. Learn. 2002, 46, 161–190.
5. Simard, P.; Steinkraus, D.; Platt, J.C. Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. In Proceedings of the 7th International Conference on Document Analysis and Recognition, Edinburgh, UK, 3–6 August 2003; Volume 2, pp. 958–963.
6. Meier, U.; Cire ̧san, D.C.; Gambardella, L.M.; Schmidhuber, J. Better Digit Recognition with a Committeeof Simple Neural Nets. In Proceedings of the 2011 International Conference on Document Analysis andRecognition, Beijing, China, 18–21 September 2011; pp. 1250–1254.
7. S.Sudhakar, V.Vijayakumar, C.SathiyaKumar, V.Priya, LogeshRavi, V.Subramaniyaswamy, Unmanned Aerial Vehicle (UAV) based Forest Fire Detection and monitoring for reducing false alarms in forest-fires, Elsevier- Computer Communications 149 (2020) 1–16, https://doi.org/10.1016/j.comcom.2019.10.007
8. M. M. A. Ghosh and A. Y. Maghari, "A Comparative Study on Handwriting Digit Recognition Using Neural Networks," 2017 International Conference on Promising Electronic Technologies (ICPET), Deir El-Balah, 2017, pp. 77-81.