A
Project-1 Report
On
"Mnist Dataset"

Prepared by
Mahesh Abburi – 110095242
Jasmin Patel- 110095248
Mohammed Farhan Baluch – 110093799

"As a student of the University of Windsor, we pledge to pursue all endeavors with honor and integrity, and will not tolerate or engage in academic or personal dishonesty. We confirm that we have not received any unauthorized assistance in preparing for or writing this assignment. We acknowledge that a mark of 0 may be assigned for copied work."

COMP 8720 (Topics: Artificial Intelligence)
Prof. Robin Gras

University of Windsor
Sept 2022

TABLE OF CONTENTS

LIST OF FIGURES

Abstract:

The report describes the use of a convolutional neural network to solve an image categorization issue. The MNIST datasets were used to evaluate the performance of the CNN model. The paper describes five distinct designs with variable convolutional layers, filter sizes, and fully linked layers. Experiments were carried out using several hyper-parameters, such as activation function, optimizer, learning rate, dropout rate, and batch size. The findings reveal that the choice of activation function, optimizer, and dropout rate influences the correctness of the results. For the MNIST dataset, all designs have accuracy more than 99%. A review of the acquired results and the literature reveals that CNN is appropriate for picture classification for the MNIST datasets.

Objective:

Handwritten Digit Recognition Using MNIST. Our task is to classify a given image of a handwritten digit into one of 10 classes representing integer values from 0 to 9, inclusively using a CNN that will be able to recognize the handwritten digits. CNN is a type of deep neural network commonly used for computer vision applications.

Dataset:

The MNIST database (Modified National Institute of Standards and Technology database) is a massive collection of handwritten digits that is often used to train image processing algorithms. The database is also commonly utilized for machine learning training and testing. It was made by "re-mixing" samples from the original NIST databases. There are 60,000 training pictures and 10,000 testing images compiled from over 250 writers in the MNIST database. Each picture is having 28*28 pixels in grayscale.

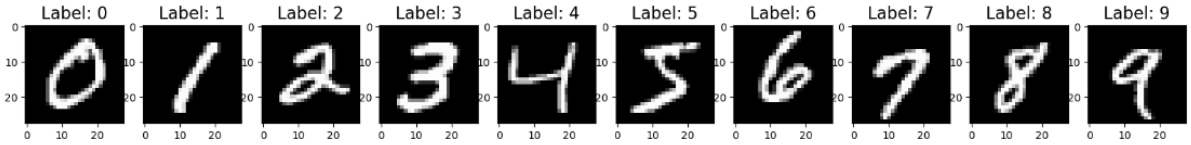

Figure 1: Data labels

Problem Summary:

Our task here is to classify a given image of a handwritten digit into one of 10 classes representing integer values from 0 to 9, inclusively.Now dataset will be loaded for training and testing. The next step is dataset preprocessing and transforming it in a way that would be convenient for our CNN model to work with. Constructing our CNN model for MNIST data training. A CNN model in general comprises of 3 layers: convolutional, pooling, and dense. The next step is fitting the dataset into the model and evaluating the model to know the accuracy.

Technology Review:

CNN is a model known as Convolutional Neural Network is a class of artificial neural networks, most applied to analyze visual imagery. An input layer, hidden layers, and an output layer comprise a convolutional neural network. Any intermediary layers in a feed-forward neural network are referred to be hidden because their inputs and outputs are veiled by the activation function and final convolution. The hidden layers of a convolutional neural network include convolutional layers. Typically, this comprises a layer that does a dot product of the convolution kernel and the input matrix of the layer. This is often the Frobenius inner product, and its activation function is typically ReLU. The convolution procedure creates a feature map as the convolution kernel slides along the input matrix for the layer, which then contributes to the input of the following layer. This is followed by further layers, such as pooling layers, fully connected layers, and normalization layers.

Development Approach:

Instead of using the pre-trained models, we developed a new convolutional neural network model from scratch. We imported the dataset using the keras API and loaded the dataset in different sets for our use cases. To estimate a model's performance for a specific training run, we further divide the training set into a train and validation dataset. Over each run, performance on the train and validation datasets may be displayed to offer learning curves and insight into how effectively a model is learning the issue. The first step is to develop a baseline model. After that we need to load dataset and apply some preprocessing. Like one hot encoding for the class element of each sample, preparing pixel data and rescaling between 0 and 1. Then to add various layers like convolutional layer with appropriate filters, max pooling layers having ReLu functions, dense layers to baseline model. We also applied batch normalization to standardize the outputs. At last, we will run and evaluate our model with testing dataset.

Hardware Requirements:

Instead Considering the average hardware requirements for the development team, the following resources will be required. The average hardware resources for our project are as follows:

1. RAM: 8 GB
2. SSD: 500 GB
3. 8 core CPU

Software Requirements:

1. IDE – Anaconda's Jupyter Notebook
2. Programming Language - Python
3. Libraries used – Keras, TensorFlow, NumPy, Matplotlib, Sklearn, Pandas

Study of the current system:

MNIST Digit Recognition dataset is one of hot favorite dataset for students beginning in the field of Machine Learning and Deep Learning. Many researchers produce their own neural nets to test results on this dataset. For instance, according to paper [1] Authors used simple convolutional neural network with three different kernel sizes along with three homogeneous ensemble networks. They got the testing accuracy of 99.87%. Similarly, In paper [2] Authors used Simplenet to create tradeoff between the computation/memory efficiency and the

accuracy. They claimed that their 13-layer architecture outperformed Keras models like VGGNEt, ResNet, and GoogleNet. So, there are thousands of researchs enrolled in this Dataset.

Code explanation:

- The first step is to import the essential tensorflow library. The dataset is then imported via the Keras API and loaded into several sets of training and testing data.
- Then we proceed to pre-process the dataset that will be used in our model. We restructure the array to have a single channel because all photos are 28*28px and grayscale. The integer values are then converted into binary vectors using one-hot encoding. Because pixel values are between 0 and 255, we rescale them to the range 0 to 1.
- We used TSNE to visualize data to 2 dimensions. Because the dataset visualization at 784 dimensions was not possible for human eye. Figure-2 is the result of plotting 1000 data point using TSNE.

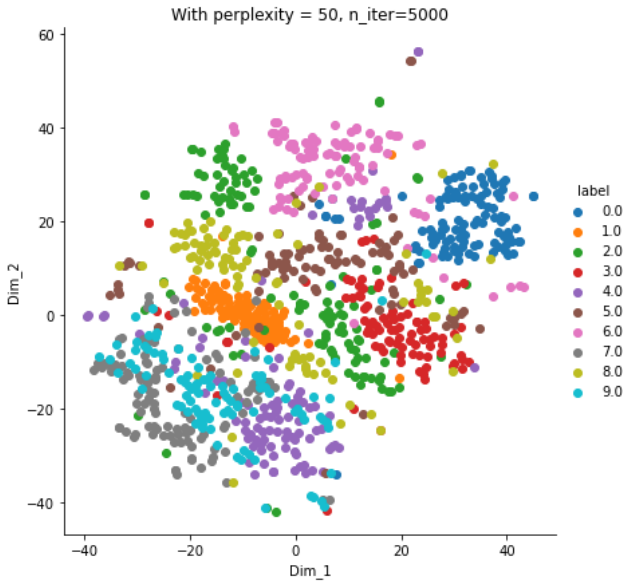

Figure 2: TSNE Data visualization in 2D

- Then we build a basic CNN model. We are employing a minimal filter kernel size of 3 for the feature extraction. The number of filters is set to 32. Then, using the Relu activation function, we boost the nonlinearity in our images. Then, by standardising the outputs, we use batch normalisation to adjust the distribution of the output layer.
- The pooling layer comes next. For the maximum pooling, we chose strides of 2 and a 2*2 matrix. The filter maps are then flattened to get features that may be fed into the CNN model.
- The entire connection dense layer is followed by a layer with 128 nodes to interpret the characteristics. Because the output has ten classes, we provide a layer with tennodes and set the activation to 'softmax' for multi-class classification. Figure-3 is the summary of our model.

```
In [18]: 1 cnn.summary()
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320

batch_normalization (BatchN  (None, 26, 26, 32)       128
ormalization)

max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
)

conv2d_1 (Conv2D)            (None, 11, 11, 32)        9248

max_pooling2d_1 (MaxPooling  (None, 5, 5, 32)         0
2D)

flatten (Flatten)            (None, 800)               0

dense (Dense)                (None, 128)               102528

batch_normalization_1 (Batc  (None, 128)              512
hNormalization)

dense_1 (Dense)              (None, 10)                1290

=================================================================
Total params: 114,026
Trainable params: 113,706
Non-trainable params: 320
_________________________________________________________________
```

Figure 3: Model Summary

- The CNN is then compiled; we use the 'adam' optimizer to train the model using stochastic gradient-descent, the categorical cross-entropy loss function, and accuracy to quantify performance.
- Following that, we train the model with 15 training epochs and provide inputs in a batch of 32 photos. And compare performance with the validation data at each epoch. Our CNN model is now ready to classify the images. We got training loss of 0.0034, training accuracy of 0.9988, validation loss of 0.0414, and validation accuracy of 0.9921 at the end of 15 epochs.

```
1 history = cnn.fit(trainX, trainY, epochs=15, batch_size=32, validation_data=(testX, testY))
Epoch 1/15
1875/1875 [==============================] - 15s 8ms/step - loss: 0.0049 - accuracy: 0.9984 - val_loss: 0.0307 - val_accuracy:
0.9925
Epoch 2/15
1875/1875 [==============================] - 22s 12ms/step - loss: 0.0064 - accuracy: 0.9979 - val_loss: 0.0363 - val_accuracy:
0.9913
Epoch 3/15
1875/1875 [==============================] - 17s 9ms/step - loss: 0.0047 - accuracy: 0.9985 - val_loss: 0.0355 - val_accuracy:
0.9907
Epoch 4/15
1875/1875 [==============================] - 18s 9ms/step - loss: 0.0050 - accuracy: 0.9985 - val_loss: 0.0356 - val_accuracy:
0.9918
Epoch 5/15
1875/1875 [==============================] - 18s 10ms/step - loss: 0.0047 - accuracy: 0.9985 - val_loss: 0.0348 - val_accuracy:
0.9918
Epoch 6/15
1875/1875 [==============================] - 18s 10ms/step - loss: 0.0042 - accuracy: 0.9985 - val_loss: 0.0349 - val_accuracy:
0.9916
Epoch 7/15
1875/1875 [==============================] - 20s 11ms/step - loss: 0.0043 - accuracy: 0.9987 - val_loss: 0.0356 - val_accuracy:
0.9921
Epoch 8/15
1875/1875 [==============================] - 19s 10ms/step - loss: 0.0035 - accuracy: 0.9988 - val_loss: 0.0368 - val_accuracy:
0.9911
Epoch 9/15
1875/1875 [==============================] - 18s 10ms/step - loss: 0.0033 - accuracy: 0.9989 - val_loss: 0.0407 - val_accuracy:
0.9912
Epoch 10/15
1875/1875 [==============================] - 23s 12ms/step - loss: 0.0042 - accuracy: 0.9986 - val_loss: 0.0361 - val_accuracy:
0.9920
Epoch 11/15
1875/1875 [==============================] - 22s 12ms/step - loss: 0.0027 - accuracy: 0.9991 - val_loss: 0.0380 - val_accuracy:
0.9908
Epoch 12/15
1875/1875 [==============================] - 19s 10ms/step - loss: 0.0042 - accuracy: 0.9985 - val_loss: 0.0467 - val_accuracy:
0.9903
Epoch 13/15
1875/1875 [==============================] - 19s 10ms/step - loss: 0.0028 - accuracy: 0.9991 - val_loss: 0.0377 - val_accuracy:
0.9935
Epoch 14/15
1875/1875 [==============================] - 19s 10ms/step - loss: 0.0028 - accuracy: 0.9990 - val_loss: 0.0488 - val_accuracy:
0.9898
Epoch 15/15
1875/1875 [==============================] - 20s 11ms/step - loss: 0.0034 - accuracy: 0.9988 - val_loss: 0.0414 - val_accuracy:
0.9921
```

Figure 4: Fitting/Training results

- At last, we used our testing data for checking accuracy of our model. By evaluation we got testing accuracy of 99.21%. We used Accuracy as our evaluation Metrix because we found out that the Dataset was unbiased. i.e., Data were almost equally divided in each category. Figure-5 is the confusion matrix of testing split. Firgure-6 and Figure-7 shows the improvement of model over time while training with 15 epochs.

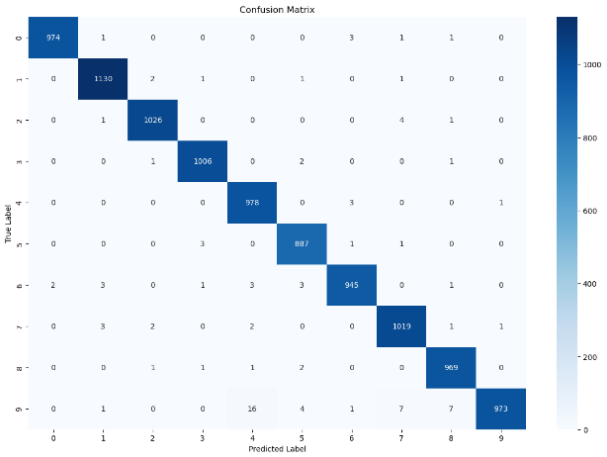

Figure 5: Confusion Matrix

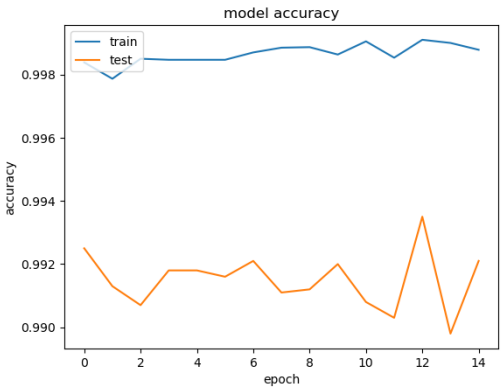

Figure 6: Model Accuracy vs Epochs

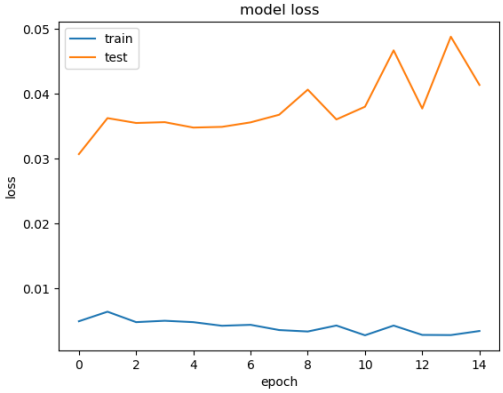

Figure 7: Model Loss vs Epochs

Comparision with existing model:

The model in link[3] "How to Develop a CNN for MNIST Handwritten Digit Classification" was created in a sequential model with a filter size of (3,3) and the ReLU activation function was utilised in all layers. Five-fold cross-validation has been used to evaluate the model. The value of k is five. This model's training outcomes are val_accuracy=0.986 and val_loss=0.05, while testing results are val_accuracy=0.978 and val_loss=0.0578.

The second known model from refernce paper[4] "Deep learning utilising rectified linear units (ReLU)" uses rectified linear units (ReLU) as a CNN classification function. ReLU is commonly employed as an activation function. The model's outcomes for training are val_accuracy=0.982 and Val_loss=0.05, while for testing, val_accuracy=0.9777 and val_loss=0.057.

When compared to the constructed model, which is similarly a sequential CNN model for mnist classification, the results for training are val_accuracy= 0.9921 and val_loss=0.041, while for

testing, the results are accuracy=0.9988 and loss=0.0034. After these models are compared, the created CNN model has somewhat higher accuracy and less loss.

Conclusion:

With the help of this project, we were able to know essence behind Deep Learning using Convolutional neural networks. We also came across with popular libraries like Keras, TensorFlow, NumPy, Matplotlib, Sklearn, Pandas. We got excellent results to determine numerical 0-9 values. The project was so much exciting for us and now we are planning to use this basic model to determine other handwritten numerical values of other languages we are known so far.

References:

[1] An, Sanghyeon et al. “An Ensemble of Simple Convolutional Neural Network Models for MNIST Digit Recognition.” ArXiv abs/2008.10400 (2020): n. pag.

[2] Hasanpour, Seyyed Hossein, Mohammad Rouhani, Mohsen Fayyaz, and Mohammad Sabokrou. “Lets Keep It Simple, Using Simple Architectures to Outperform Deeper and More Complex Architectures.” arXiv.org, February 14, 2018. https://arxiv.org/abs/1608.06037v7.

[3] https://machinelearningmastery.com/save-load-keras-deep-learning-models/

[4] Agarap, A.F. (2019) Deep learning using rectified linear units (ReLU), arXiv.org. Available at: https://arxiv.org/abs/1803.08375 (Accessed: October 17, 2022).