PATTERN RECOGNITION OF HANDWRITTEN DIGITS MNIST DATASET

Meshaal. Mouawad
Department of Electrical and Computer Engineering, Mississippi State University
This technical report is to satisfy the requirement of Pattern Recognition course Spring, 2021.
Project Report

Abstract—. **MNIST, Modified National Institute of Standards and Technology, is the largest database of handwritten numbers used in deep learning, and machine learning. In this project, a hands-on experience of applying machine learning and pattern recognition techniques is given to a real-world data set such as MNIST. Multiple building blocks have been proposed and analyzed to improve the speed and the accuracy of the Convolutional Neural Networks (CNN). Two networks have been used with the same data. In network I, a three-layer MLP with ReLU and dropout resulting in fast training process with over all accuracy 95% during training and 94% for testing. Network II on the other hand, a stack of CNN, RelU, and Max pooling shows slower training process with better accuracy than network I and overall, 99% accuracy for training, and 98.9% for testing. Another modification on network II improved overall accuracy during the training to 99.82% and accuracy for testing to 99.25%. this modification will be shown in the report. The building blocks for the project will be discussed briefly with the results and figures. Python code is also provided for this project. This project may be used for as a guidance for new students or engineers who aiming to understand pattern recognition.**

Index Terms: **Pattern Recognition, Convolutional Neural Networks, Deep learning, MNIST.**

I. INTRODUCTION

MNIST stands for *Modified National Institute of Standards and Technology*. MNIST is a database of handwritten digits that used for training and testing many image processing systems and machine learning research [2]. It has 60,000 training images and 10,000 testing images of handwritten digits and characters [3]. Later in 2017, an extended database has been published that has 240,000 training images and 40,000 testing images. Since MNIST is a standard training dataset for digits in English, in recent times, others were also provided similar databases for training digit datasets in other languages. The dataset is considered as a benchmark for machine learning worldwide. MNSIT database is good for people who want to try machine learning techniques and *pattern recognition* methods on real-world data while spending minimal efforts on preprocessing and formatting [2]. It reduces the time and effort that spend on preprocessing and formatting of data. an example of MNIST dataset is shown in Fig. 1.

Keras deep learning library, a popular deep learning library, is implemented in this project. Google's TensorFlow, a popular opened source deep learning library, uses Keras as a high-level API for its library [4] and it is easy to be used and learned. There are two networks were used in this project from opened source GitHub [4].

In this project, the MNIST dataset was used, then neural layers have been applied in the building blocks such as convolution layer, max pooling, flatten, dropout, dense, and activation layer. Each layer has its duties to perform. They improve both the speed and accuracy of the convolutional neural networks (CNN). In this part, the layers used in this project would be explained shortly. While The dataset was already given the option of the loss function, the optimizer, and the regularizer, is must be determined such that the model can be trained. For the optimization, an Adaptive Moments (Adam) algorithm were used.

This Report is organized as follows. In Section II the deep learning artificial neural networks is described, and the type of layers throughout the project are explained briefly. In Section III, the MNIST digit classifier model is introduced, and the proposed Multilayer Perceptron (MLP) for digit classification explained. In Section IV, the approaching method for pattern recognition with neural network is explained in detail. The results & discussion is described in Section V, where two networks were used, and the results shown in this section from Python window. In Section VI. compering with another classification methods were listed with some details. In Section VII, the conclusion is drawn. The code for this project is provided in the appendix.

II. DEEP LEARNING ARTIFICIAL NEURAL NETWORKS

An artificial neural network represents the structure of a human brain modeled on the computer vision. It consists of neurons

Meshaal Mouawad, PhD Candidate, Mississippi State University, Starkville, MS 39759 USA (e-mail: mm4922@msstate.edu).
This report was submitted on April 30th, 2021, for Course project to satisfy the requirements of ECE 8440 Pattern Recognition course, PhD level course, to Dr. Tang Bo. Assistant Professor at Mississippi State University.

* Some of the information in this report was part of [1] M. M. NABI, F. Islam, M. M. Farhad, and M. Mouawad, "Digital Signal Processing Mini Project I Report," Mississippi State University 2020.
The information was used mostly from “Advanced Deep Learning with TensorFlow 2 and Keras - Second Edition,” *Packt*. [Online]. Available: https://www.packtpub.com/product/advanced-deep-learning-with-tensorflow-2-and-keras-second-edition/9781838821654. [Accessed: 13-Apr-2021].

and synapses, weights, biases, and functions organized into layer [5]. The main architectures of deep learning:

1) *Convolutional neural networks (CNN).*
2) *Recurrent neural networks (RNN).*
3) *Generative adversarial networks (GAN).*
4) *Recursive neural networks.*

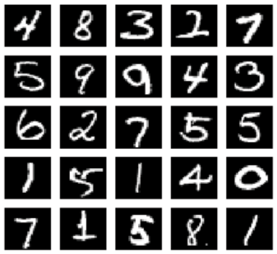


Fig. 1.: Example images from the MNIST dataset. Each grayscale image is 28 × 28-pixels.

In this project, the layers and the functions were used are convolutional layer, max pooling, flatten, dropout, dense, and fully connected layers.

B. *Convolutional layer*

A convolutional layer includes a series of filters that need to be taught about the parameters. The filter height and weight are less than the size of the input. To compute an activation map made of neurons, each filter is transformed with the volume of inputs. In computer vision, one of the difficulties is that images can be very large and thus computationally costly to work on. For practical uses, we need quicker and computationally cheaper algorithms. A simple neural network that is completely connected will not help. Fig. 2. shows the CNN model for the MNIST digit classification. In Kernel, the `Conv2D` is the parameter of the convolution layer.

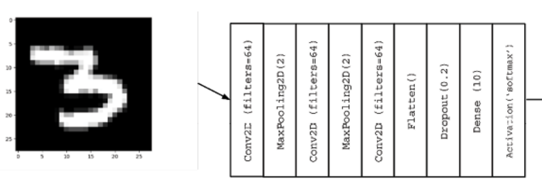


Fig. 2.. The CNN model for MNIST digit classification [6]

The kernel can be visualized as a rectangular patch or window that slides through the whole image from left to right, and from top to bottom as it shown in Fig. 3. [4].

C. *Max pooling*

Max pooling is a form of operation usually applied to individual convolution layers of CNNs. The key idea behind a layer of pooling is to "accumulate" characteristics from maps created by converting a filter over an image. Formally, its purpose is to gradually reduce the representation's spatial size to reduce the number of parameters and computation within the network. Max Pooling is the most common method of pooling. Max pooling decreases the dimensionality of images when applied to a model by reducing the number of pixels in the output from the preceding convolution layer. Fig. 4. explained the maxpooling layer. In Kernel, we use `MaxPooling2D` with the argument `pool_size=2`. `MaxPooling2D` compresses each feature map. Every patch

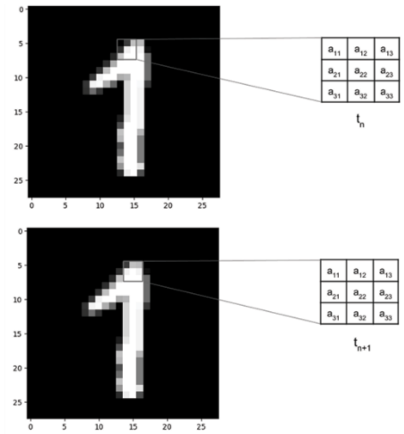


Fig. 3. A 3 × 3 kernel is convolved with an MNIST digit image [6].

of size `pool_size × pool_size` is reduced to 1 feature map point. Then the maximum feature point value within the patch is equal to the value [4].

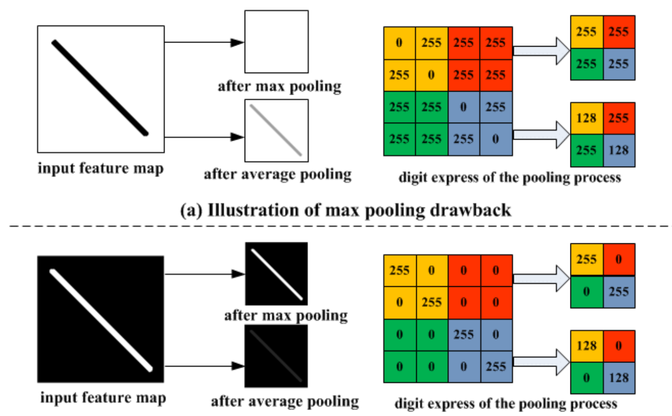


Fig. 4. An example for Max pooling operation [7].

D. *Flatten*

Flatten is used to flatten the input. For example, if flatten is applied to a layer having input shape as `(batch_size, 1,1)` then the output shape of the layer will be `(batch size, 2)`. Kernel uses `flatten` parameter to apply flatten layer. Flattening occurs when a reduction of all layers to one background layer is needed. Layers will increase the size of files, while also tying up available resources for processing. Anyone can choose to combine some layers or even flatten the whole image to one background layer to keep down the file size.

E. *Dense*

A regular deeply connected neural network layer is a `dense` layer. The `dense` layer performs the input operation below and returns the output. The dense layer is also known as a fully connected layer (FC). Fully connected or dense layers are those layers in a neural network where all inputs from one layer are connected to each activation unit of the next layer. The last few layers are fully connected layers in most common machine learning models that compile the data extracted from previous layers to form the final output. In Fig. 5. It shows the structure of `dense` layer (Fully Connected).

F. Dropout

A single model can be used by randomly dropping out nodes during training to simulate many distinct network architectures. This is called `dropout` and provides a highly computationally cheap and surprisingly efficient method of regularization to minimize overfitting and increase generalization error in all forms of deep neural networks. The `Dropout` layer at random sets input units to 0 at each stage during the training period with a rated frequency that helps prevent overfitting. Inputs not set to 0 are scaled to 1/(1-rate) such that there is no difference in the sum for all inputs.

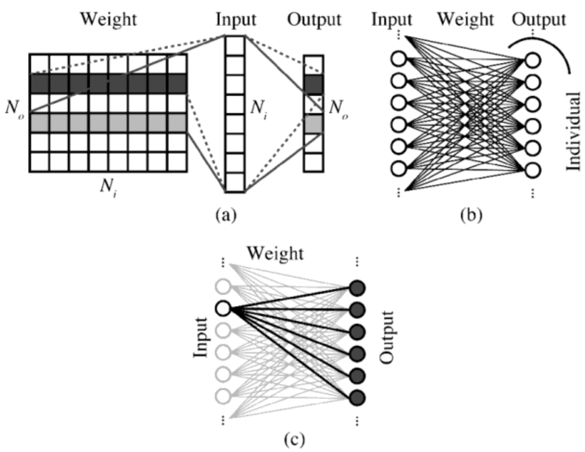


Fig. 5. The structure of the fully connected layer and the data dependency [8].

G. Activation function

In neural networks, a node's activation function determines the node's output given an input or set of inputs. Depending on the input, a typical integrated circuit is a digital network of activation functions that can be 'ON' (1) or 'OFF' (0). One linear function enables each layer. In turn, this activation goes as an input into the next step and the second layer calculates the weighted sum on that input and, in turn, fires based on another feature of linear activation. There some common activation functions such as `ReLU`, `selu`, and, `sigmoid`. Dense layer can also be used as activation function but since the MNIST digit classification is a non-linear process, we need another activation function.

`ReLU` activation function works by allowing only positive inputs to pass through unchanged and blocked any others. The mathematical structure of ReLU function is described in Equation (1)

$$relu(x) = max(0, x) \tag{1}$$

A common activation function is the `sigmoid` activation function and the mathematical expression in shown in Equation (2)

$$sigmoid(x) = \frac{1}{1-e^{-x}} \tag{2}$$

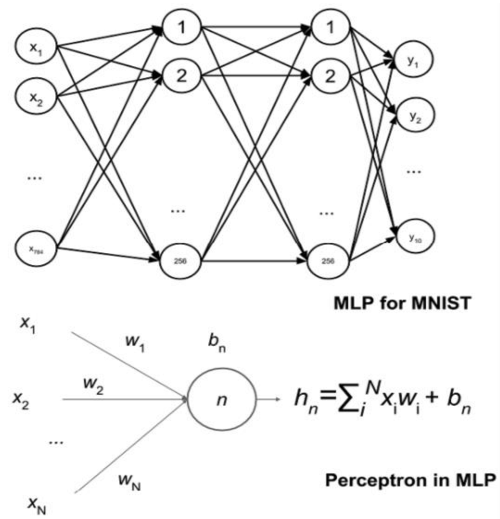


Fig. 7. The MLP MNIST digit classifier is made of fully connected layers. For simplicity, the activation and dropout layers are not shown. One unit or perceptron is also shown in detail

III. The MNIST Digit Classifier Model

The classifier model is implemented using the Sequential API of Keras in this project with Python.3. The proposed MLP model shown in Fig. 6. can explain and used for MNIST digit Classification. When the units or perceptron are exposed, the MLP model is a fully connected network, as shown in Fig. 7. The output of the perceptron is computed from inputs as a function of weights, W_i, and bias, b_n, for the *n-th* unit.

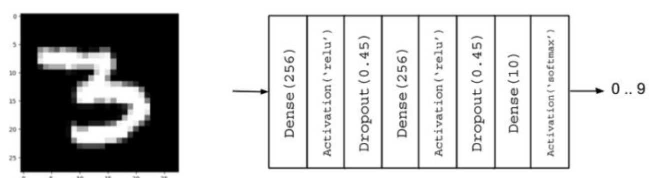


Fig. 6. The MLP MNIST digit classifier model [6]

IV. Approaching Methods

`Keras` deep learning library, a popular deep learning library with over 370,000 developers using it at the time of writing – a number that is increasing by about 35% every year is implemented in this project. Google's TensorFlow, a popular opened source deep learning library, uses Keras as a high-level API for its library [4]. Keras is used by Google, Netflix, Uber, and NVIDIA [4]. There are two networks were used in this project from opened source GitHub [4]. Since the dataset were already given the choice of the loss function, the optimizer, and the regularizer, should be determined that the model can be trained. For optimization, an Adaptive Moments [(`Adam`] algorithm was used. The MNIST dataset for hand-written digits is fed into two neural networks. Each network will be explained in detail.

1. Network I:

This network consists of a 3-layer multilayer perceptron (MLP) network where it has `ReLU` activation and `dropout` after each layer. `ReLU` sets all non-positive values in the matrix to zero, and rest of the values are remained steady. `dropout` reduces overfitting in neural networks and precludes complicated co-alterations on training data. The model is sequential. The `batch_size` is set to `128`, `hidden_unit` is set to `32`, and `dropout` is `0.45` initially. The network is tested with changing different parameters. To get the output with a one-hot encoded vector, the `SoftMax` activation function is used. The `SoftMax` function uses an input vector and stabilizes it into a likelihood division. The network architecture is presented briefly in Fig. 8. In this sequential neural network, the `input_size` is `28 X 28` grayscale image (channel 1). This input is fed to a dense layer and the `output_size` is `32`. The total number of parameters used in this layer is 25,120 parameters and calculated using Equation (3)

$$P_r = K((L \times M \times N) + 1) \tag{3}$$

Where M is the width, N is the height, L is the previous layer's filters, K is the current layer filter, and 1 is the bias term.

$$P_r = 25{,}120 + 1{,}056 + 330 = 26{,}506\, prameters$$

After dropping out, number of parameters used in the next `dense` layer by Equation (3) is 1,056 parameters. Once again after dropping out from the `ReLU` activation layer, lastly, we get

TABLE I
THE COMPONENTS OF NETWORK I

Layer Type	No. Parameters	Output Size
Dense	25120	32
Activation	0	32
Dropout	0	32
Dense 1	1056	32
Activation 1	0	32
Dropout 1	0	32
Dense 2	330	10
Activation 2	0	10
Total layers: 3		
Total Components:	26506	

Summery Summary of MNIST digit classifier model from Python

10 output using the `SoftMax` layer, then the total number of parameters used here is 330. Table I. shows each component used in Network I, their parameter number, and `output size`. The Total Model summary is illustrated in Listing. I. The total trained parameters for Network I are 26,506 parameters. In Network I, `Adam` optimizer is used. The loss function of this classifier is set Categorical Cross Entropy `categorical_crossentropy`. The Category Cross Entropy can be calculated by Equation (4)

$$-\sum_{i=1}^{categoryies} y_i^{label} \log y_i^{prediction} \tag{4}$$

The model with `20` epochs was trained initially. In this network, the training process is much faster. It takes 1.0 second per epoch. In these network settings, the training accuracy was 95% and a test accuracy was 94%. In Section V, the discussion of overall network accuracy is described in detail.

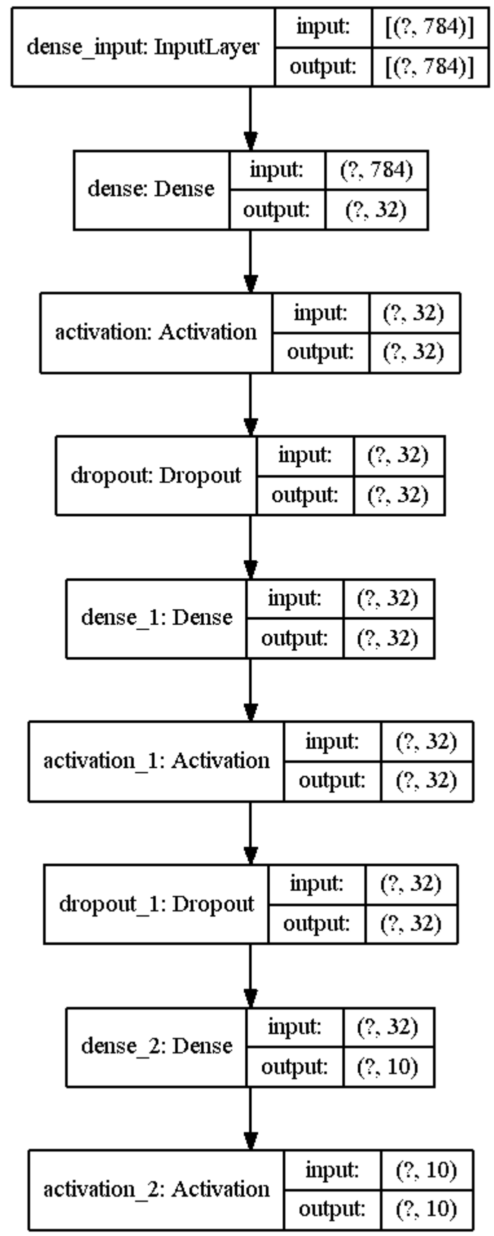


Fig. 8. Network architecture of Network I of MNIST dataset (mp1_nn1.py)

1. Network II:

A stack of CNN-`ReLU` `Maxpooling` model is established to test the network for the same dataset. It is a combination of a 3-layer 2D convolutional layer. It has a `MaxPooling2D` layer and a `dropout` layer. A dataset of `28 X 28` grayscale image is fed into the network. Initially, the `batch_size` is set to `128`. The `kernel_size` is `3` and the `pool_size` is `2`. In the beginning, the filter size was `16`, and the `dropout` was `0.3`. The output layer is a 10-dimension one-hot vector, which can be done by using the `SoftMax` layer. `Regularization` was implemented by adding a dropout layer. The total network architecture is presented in Fig. 9. The total number of parameters used for Network II is 6,250 parameters. Total Model summary is illustrated in Listing II. The total trained parameters for

Network II is 6,250 parameters.

LISTING I
THE MODEL SUMMARY OF NETWORK I

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 32)                25120
_________________________________________________________________
activation (Activation)      (None, 32)                0
_________________________________________________________________
dropout (Dropout)            (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                1056
_________________________________________________________________
activation_1 (Activation)    (None, 32)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                330
_________________________________________________________________
activation_2 (Activation)    (None, 10)                0
=================================================================
Total params: 26,506
Trainable params: 26,506
Non-trainable params: 0
```

Summery output of network I from Python. Python 3 with TensorFlow 2 is used in this project.

In this sequential neural network, the `INPUT_SIZE` is `28 X 28` gray scale image (channel 1). This input is fed into a `conv2D`

TABLE III
THE TRAINING PARAMETERS MODIFICATION OF NETWORK II

Parameters	Changed Value
Batch size	64
Kernel size	3
Filters	64
Dropout	0.3

Summary of the modification of the network II for better performance

layer where is `output_size` is `28 X 28`. Total number of parameters is used in this layer is 160. After `MaxPooling2D`, the `Kernel_size` becomes half, after the next dense layer, the number of parameters used in the next dense layer was 2,320 parameters. Once again, after applying the maxpooling and `Conv2D`, the total number of used parameters in this layer was 2,320 parameters. Lastly, applying flattening, the size of the

TABLE II
THE COMPONENTS OF NETWORK II

Layer Type	No. Parameters	Output Size
Max_pooling2d	0	32
Conv2d_1	2320	32
Max_pooling_1	0	32
Conv2d_2	2320	32
Flatten	0	32
Dropout	0	10
Dense	1450	10
Activation	0	10
Total layers: 6		
Total	6250	

Summery Summary of MNIST digit classifier model from Python

kernel became `144`. Then, dropping out, the total is 1,450 parameters. Table II. shows each component used in the Network II, their parameter number, and output size.

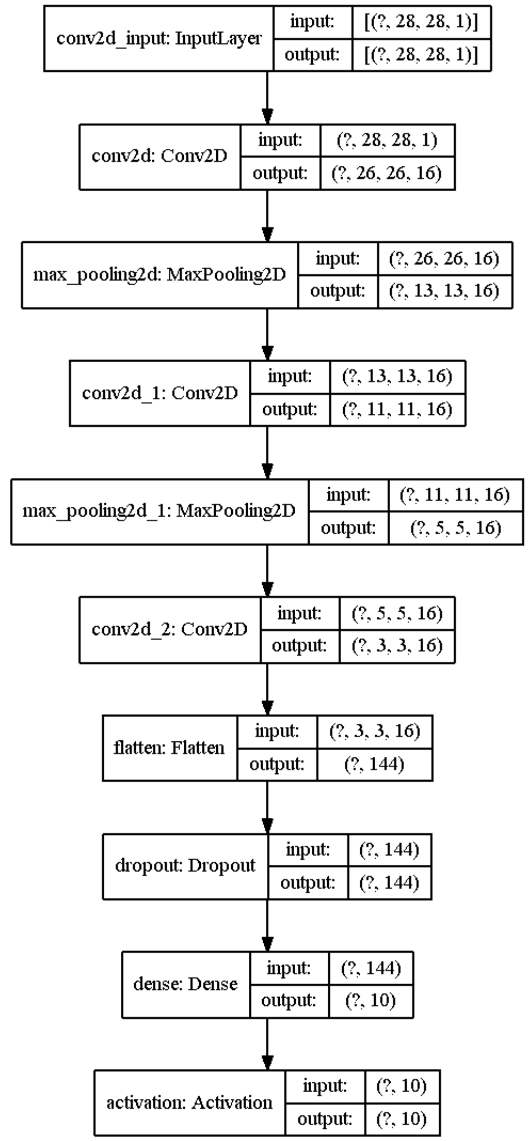


Fig. 9. Network architecture of Network II of MNIST dataset.

In Network II, `Adam` is used as an optimizer. The loss function of this classifier is set as `categorical_crossentropy`. The model was trained with `10` epochs initially. In this network, the training process is a bit slower than the Network I. It takes an average of 11 seconds per epoch. In these network settings, the results of a training accuracy were 99.023% and a test accuracy of 98.9%. In Section V, the discussion of overall network accuracy is described in detail.

A modification of the Network II was done to get better performance. The parameters were changed such as `batch_size`, `kernel_size`, `pool_size`, `filters`, and `dropout`. Moreover, the optimizer SGD (stochastic gradient descent) [`SGDClassifier`] was changed instead of

LISTING II
THE MODEL SUMMARY OF NETWORK II

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 26, 26, 16)        160
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 13, 13, 16)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 11, 11, 16)        2320
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 5, 5, 16)          0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 3, 3, 16)          2320
_________________________________________________________________
flatten_2 (Flatten)          (None, 144)               0
_________________________________________________________________
dropout_4 (Dropout)          (None, 144)               0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1450
_________________________________________________________________
activation_5 (Activation)    (None, 10)                0
=================================================================
Total params: 6,250
Trainable params: 6,250
Non-trainable params: 0
```

Summery output of network II from Python. Python 3 with TensorFlow 2 is used in this project.

`Adam`. After several trial and error, results an accuracy of 99.82% during the training set and 99.25 for the testing set. Though it took near about 50 seconds per epoch during the training sessions, optimizer 'Adam' gave us a good result according to testing. The total parameter used for this training process is presented in Listing III. The training parameters that have been used for the modified network is illustrated in Table III.

LISTING III
THE MODEL SUMMARY OF THE MODIFIED NETWORK II

```
Model: "sequential"

Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 64)        640

max_pooling2d (MaxPooling2D) (None, 13, 13, 64)        0

conv2d_1 (Conv2D)            (None, 11, 11, 64)        36928

max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0

conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928

flatten (Flatten)            (None, 576)               0

dropout (Dropout)            (None, 576)               0

dense (Dense)                (None, 10)                5770

activation (Activation)      (None, 10)                0
=================================================================
Total params: 80,266
Trainable params: 80,266
Non-trainable params: 0
```

Summery output of modified Network II. from Python. Python 3 with TensorFlow 2 is used in this project.

V. THE RESULTS & DISCUSSION

In testing the networks, the same dataset was trained with different network settings. The discussion of the results is briefly explained in this section.

A. *Network I*

For the first network, we have trained the model using `10` epochs. The Confusion Matrix for Network I for both training and testing is illustrated in Listings IV and V.

LISTING. IV
THE CONFUSION MATRIX OF NETWORK I (TRAINING)

Train Confusion Matrix:

ROWS ARE TRUE CLASSES

5804	1	8	2	8	1	38	2	58	1
1	6575	35	26	7	1	2	16	64	15
22	21	5645	68	37	10	38	43	70	4
10	7	111	5753	1	82	7	54	62	44
3	13	18	1	5631	3	61	2	12	98
35	6	18	187	31	4871	96	2	138	37
27	9	7	0	26	45	5789	0	15	0
9	16	38	26	32	6	2	6050	4	82
20	52	35	51	40	53	37	5	5513	45
15	7	0	57	219	38	6	80	46	5481

ROWS ARE TRUE CLASSES

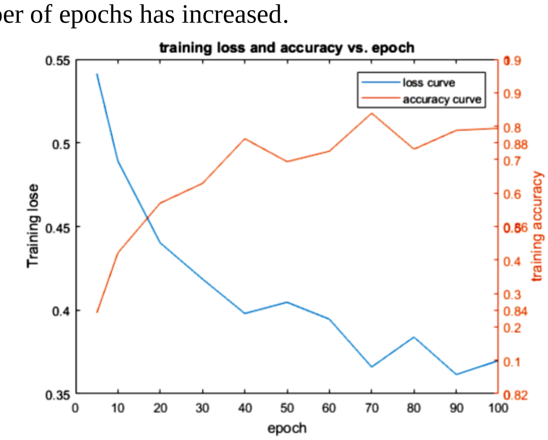
```
Confusion matrix Train stats:
Total instances : 60000
Overall accuracy (OA) : 0.95187
```

Summery output of the confusion Matrix for Network I from Python. Python 3 with TensorFlow 2 is used in this project.

From the confusion matrix, we can easily understand how the classifier can be able to classify. In the confusion matrix, the rows represent true class, and columns represent predictions. For training total, 60,000 instances are taken, and for testing 10,000. The overall accuracy can be calculated using Equation (5).

$$\frac{TP+TN}{TP+TN+FP+FN} \tag{5}$$

Where TP is the true positive, TN is true negative, FP is false positive, and FN is false negative. The overall accuracy for training is 0.95603 or 95.603% for [`10 epochs`]. The overall accuracy for testing is 0.094790 or 94.7090%
We tried to see how the number of epochs changes the loss and accuracy. From the Fig. 10. we can see that training loss has decreased when the number of epochs has increased. On the contrary, the accuracy of the training has increased when the number of epochs has increased.



LISTING. VII
THE CONFUSION MATRIX OF NETWORK II (TRAINING)

```
ROWS ARE TRUE CLASS

5881   1    13  0   0   3   9   0   14  2
1    6676   22  3   4   3   3   23  5   2
2    5    5896  16  1   1   1   14  20  2
2    2    21  6032  0   47  0   9   11  7
6    4    13  1   5698  4   18  6   23  69
0    0    4   5   0   5386  10  0   10  6
5    2    4   0   3   35  5852  0   17  0
1    4    46  6   7   3   0   6167  8   23
6    6    16  9   7   33  2   2   5748  22
16   2    0   16  14  33  1   21  12  5834
                                    COLUMNS PREDICTIONS

Confusion  matrix  Train  stats:
Confusion matrix Train stats:
Total instances : 60000
Overall accuracy  (OA)  :  0.98617
```

Summery output of the confusion Matrix for Network II from Python. Python 3 with TensorFlow 2 is used in this project.

Fig. 10. training loss and accuracy vs. epoch (Network I).

Sometimes it can be seen that the accuracy has increased, and the loss has also increased, which is a sign of overfitting the model. In Fig. 11. we can see that around epoch 60 to 80, the network prone to become overfitted.

TABLE V.
RELATION BETWEEN HIDDEN UNIT AND ACCURACY (TRAINING AND TESTING)

Hidden unit	Accuracy Training	Accuracy testing
32	0.9557	0.9447
64	0.97747	0.9667
96	0.98663	0.974
128	0.9906	0.9777
196	0.99482	0.9794
256	0.99637	0.983
512	0.9978	0.9852
1024	**0.99818**	**0.9854**

Summary of the relation between hidden unit and accuracy (training and testing)

A modification and testing of the network by changing the `hidden_unit` of the network. It is visible from Fig. 11. that when the number of hidden units has increased, the accuracy has increased too. For Network I, the accuracy becomes stable after 500 hidden unit [`hidden_unit=500`]. It is also clearly visible from the plot that training accuracy is higher than the testing accuracy, which is feasible.

B. Network II

For the first network, the model was trained using `10 epochs` initially. The Confusion Matrix for Network II is presented in Listing. V, for both training and testing. From the confusion matrix, we can easily understand how the classifier can class the prediction. We can also see there are some miss-classify the prediction. Here in the *confusion matrix*, rows represent true class, and columns represent predictions. A total of 60000 instances are taken for training, and 10000 for testing.

The overall accuracy for training is 0.99023 or 99.023% [for 10 epochs]. The overall accuracy for testing is .98900 or 98.9%

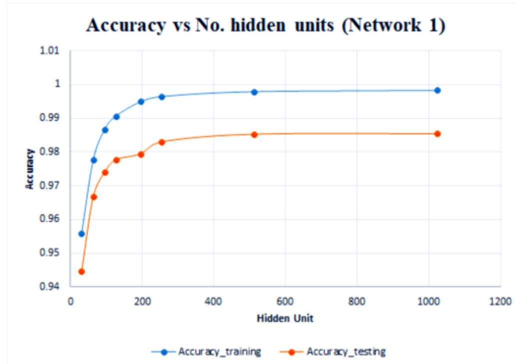


Fig. 11.: Accuracy vs No. hidden units (Network I)

The observation of the loss and accuracy by varying the number of epochs. From the Fig. 12, we can see that training loss has smoothly decreased when the number of epochs has increased. On the contrary, the accuracy of the training has gradually increased when the number of epochs has increased as well.

C. Network II (Modified)

A modification was applied to Network II for the same dataset to get better accuracy. As discussed in the approaching methods section, we have modified different parameters to get better performance. Therefore, the confusion matrix was generated. As earlier, the model has been trained with 60000 instances and got an accuracy of 0.99820 or 99.82%. the network was tested

LISTING. VI
THE CONFUSION MATRIX OF NETWORK II (TRAINING)

```
ROWS ARE TRUE CLASS.

5881   1    13  0   0   3   9   0   14  2
5881   1    13  0   0   3   9   0   14  2
1    6676   22  3   4   3   3   23  5   2
2    5    5896  16  1   1   1   14  20  2
2    2    21  6032  0   47  0   9   11  7
6    4    13  1   5698  4   18  6   23  69
0    0    4   5   0   5386  10  0   10  6
5    2    4   0   3   35  5852  0   17  0
1    4    46  6   7   3   0   6167  8   23
6    6    16  9   7   33  2   2   5748  22
16   2    0   16  14  33  1   21  12  5834
                                    COLUMNS PREDICTIONS

Confusion matrix Train stats:
Total instances : 60000
Overall accuracy (OA) : 0.98617
```

Summery output of the confusion Matrix for Network II from Python. Python 3 with TensorFlow 2 is used in this project.

to classify the image with an accuracy of 0.9925 or 99.25%, which is pretty good results. Consideration was taken to keep in mind that the network would not become overfitted in case the model was trained for a long period. We have checked it for `10 epochs`. On average, each epoch took 38 seconds, which is reasonable. The number of `filters` used for this case is `64`.

We have added a decent amount of `dropout`. We also tried `stochastic gradient descent` (SGD), which did not perform well comparing to the `Adam` optimizer. The results are shown in Listing. VIII and Listing. IX.

LISTING. V
THE CONFUSION MATRIX OF NETWORK I (TESTING)

```
          Confusion matrix Train stats:

                    ROWS ARE TRUE CLASSES
     960    0  0    1   0    4    9    3   3    0
     0   1109 3    4   0    1    4    0  13    1
     6    2  974  10   5    1    9   10  14    1
     0    0   21 957   0   12    0    8   9    3
     2    1   3    0 942    0   10    2   4   18
     9    1   2   33   7  788   19    3  16   14
     11   3   1    0   6    7  929    0   1    0
     1    6  17   10   6    1    0  972   0   15
     6    6   6    7   9   10   13    4 901   12
     6    4   0   10  48   11    3   10   7  910
                                                   ROWS ARE TRUE CLASSES

 Confusion matrix Test stats:
 Total instances : 10000
 Overall accuracy (OA) : 0.98617
```

Summery output of the confusion Matrix for Network I from Python. Python 3 with TensorFlow 2 is used in this project.

LISTING. VIII
THE CONFUSION MATRIX OF MODIFIED NETWORK (TRAINING)

```
      Test Confusion Matrix:

        ROWS ARE TRUE CLASS

      5917 0 3 0 0 1 0 1 0 1
      0 6732 0 0 0 0 0 9 0 1
      0 2 5948 0 0 0 0 7 0 1
      0 0 2 6112 0 8 0 4 3 2
      0 2 0 0 5807 0 0 1 0 32
      0 1 0 3 0 5415 1 0 0 1
      2 3 0 0 4 1 5906 0 2 0
      0 3 1 0 0 0 0 6260 0 1
      0 2 1 0 2 0 2 1 5835 8
      0 1 0 0 1 0 0 6 0 5941
                                  COLUMNS PREDICTIONS
      Confusion matrix Test
      stats:
      Total instances : 60000
      Overall accuracy (OA) : 0.99788
```

Summery output of the confusion Matrix for modified Network from Python. Python 3 with TensorFlow 2 is used in this project.

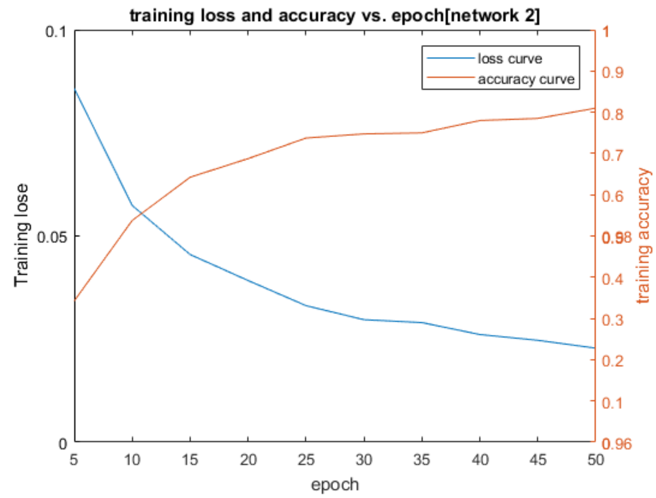


Fig. 12: training loss and accuracy vs. epoch (Network II)

LISTING. IX
THE CONFUSION MATRIX OF MODIFIED NETWORK (TRAINING)

```
      Train Confusion Matrix:

        ROWS ARE TRUE CLASS

       5917 0 3 0 0 1 0 1 0 1
       0 6732 0 0 0 0 0 9 0 1
       0 2 5948 0 0 0 0 7 0 1
       0 0 2 6112 0 8 0 4 3 2
      0 2 0 0 5807 0 0 1 0 32
       0 1 0 3 0 5415 1 0 0 1
       2 3 0 0 4 1 5906 0 2 0
       0 3 1 0 0 0 0 6260 0 1
       0 2 1 0 2 0 2 1 5835 8
       0 1 0 0 1 0 0 6 0 5941
                                   COLUMNS PREDICTIONS

      Confusion matrix Train stats:
      Total instances : 60000
      Overall accuracy (OA) : 0.99788
```

Summery output of the confusion Matrix for Modified Network from Python. Python 3 with TensorFlow 2 is used in this project.

VI. COMPERING ANOTHER CLASSIFICATION METHOD

With some classification methods (particularly template-based methods, such as SVM and K-nearest neighbors), the error rate improves when the digits are centered by bounding box rather than center of mass [1]. Table 5 shows examples of some methods and the results [1].

TABLE. VI.
EXAMPLES OF OTHER METHODS TRAINED & TESTED

classifier	Preprocessing	Error Rate (%)
linear classifier (1-layer NN)	none	12.0
linear classifier (1-layer NN)	deskewing	8.4
pairwise linear classifier	deskewing	7.6
K-nearest-neighbors, Euclidean (L2)	none	3.09
K-nearest-neighbors, L3	none	2.83

K-NN, Tangent Distance	subsampling to 16x16 pixels	1.1
SVM, Gaussian Kernel	none	1.4
SVM deg 4 polynomial	deskewing	1.1
Reduced Set SVM deg 5 polynomial	deskewing	1.0
Virtual SVM deg-9 poly [distortions]	none	0.8

VII. Conclusion

The MNIST dataset for handwritten digit recognition is analyzed in this project. For the same dataset, two different networks are used. In the first network, a 3-layer MLP with ReLU and dropout is used after each layer. The training process is fast in these network settings. An overall accuracy of 95% during training and 94% for testing were recorded. In Network II a stack of CNN, RelU, and Maxpooling is used. The training process is a little bit slower, but it gives better accuracy than the Network I. We got 99% overall accuracy for training, and 98.9% for testing. We have also modified the Network II to get better accuracy. We have added more filter, reduced the batch size, and added a 30% dropout. This modified Network II gave us 99.82% overall accuracy during the training and 99.25% accuracy for testing. During the training period, the loss where investigated when the number of epochs has increased as it might have caused overfitting of the data. In conclusion, the Network II gives better performance in this project than Network I for the MNIST dataset for handwritten digit pattern recognition.

VIII. References

[1] M. M. NABI, F. Islam, M. M. Farhad, and M. Mouawad, "Digital Signal Processing Mini Project I Report," Mississippi State University 2020.

[2] Y. LeCun. "The MNIST database of handwritten digits." http://yann.lecun.com/exdb/mnist/ (accessed.

[3] E. Kussul and T. Baidyk, "Improved method of handwritten digit recognition tested on MNIST database," *Image and Vision Computing,* vol. 22, pp. 971-981, 10/01 2004, doi: 10.1016/j.imavis.2004.03.008.

[4] B. R. Atienza, "Advanced Deep Learning with TensorFlow 2 and Keras - Second Edition," second edition ed: Packt Publishing Limited, pp. Fig. 1. Example images from the MNIST dataset. Each grayscale image is 28 × 28-pixels.

[5] A. Khvostikov, K. Aderghal, J. Benois-Pineau, A. Krylov, and G. Catheline, "3D CNN-based classification using sMRI and MD-DTI images for Alzheimer disease studies," *arXiv pre-print server*, 2018-01-18 2018, doi: None
arxiv:1801.05968.

[6] PerceptiLab, "Fig. 1- Fig. 3 & Fig. 6- Fig. 7. Advanced Deep Learning with TensorFlow 2 and Keras," in *PerceptiLabs*, Second Edition ed: Packt Publishing Limited.

[7] S. Khan, M. Hayat, and F. Porikli, "Regularization of Deep Neural Networks with Spectral Dropout," p. arXiv:1711.08591. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2017arXiv171108591K

[8] M. Gottschling and R.-E. Irimia, "Taxonomic revision of Rochefortia Sw. (Ehretiaceae, Boraginales)," *Biodiversity Data Journal*, vol. 4, p. e7720, 2016, doi: 10.3897/bdj.4.e7720.

[9] Y. Gavrilova. "A Guide to Deep Learning and Neural Networks " Serokell. https://serokell.io/blog/deep-learning-and-neural-network-guide (accessed 2021).

[10] B. Kayalibay, G. Jensen, and P. V. D. Smagt, "CNN-based Segmentation of Medical Imaging Data," *ArXiv,* vol. abs/1701.03056, 2017.

[11] Y. Wang, H. Li, P. Jia, G. Zhang, T. Wang, and X. Hao, "Multi-Scale DenseNets-Based Aircraft Detection from Remote Sensing Images," *Sensors*, vol. 19, p. 5270, 11/29 2019, doi: 10.3390/s19235270.

[12] R. Wei, F. Zhou, B. Liu, B. Liang, B. Guo, and X. Xu, "A CNN based volumetric imaging method with single X-ray projection," in *2017 IEEE International Conference on Imaging Systems and Techniques (IST)*, 18-20 Oct. 2017 2017, pp. 1-6, doi: 10.1109/IST.2017.8261550. [Online]. Available: https://ieeexplore.ieee.org/document/8261550/

[13] H. Wu and X. Gu, "Max-Pooling Dropout for Regularization of Convolutional Neural Networks," 12/04 2015.

[1-13]

APPENDIX. I

The project code was written in Python 3 programing language. There are some other packages, libraries, and API that were used in this code. the code was run on python environment using Colaboratory Notebook (Colab), an interactive environment provided by Google. Colab Notebook can be access from Welcome To Colaboratory - Colaboratory (google.com)

What is the Colaboratory?

Colaboratory, or "Colab" for short, allows you to write and ex ecute Python in your browser, with

- Zero configuration required
- Free access to GPUs
- Easy sharing

Whether you're a student, a data scientistor an AI researcher, Colab can make your work easier.

In **Section A**. Installing the packages needed for this project or if anyone want to learn it is provided by Advanced Deep Learning with TensorFlow 2 and Keras - Second Edition," *Packt*. [Online]. Available: https://www.packtpub.com/product/advanced-deep-learning-with-tensorflow-2-and-keras-second-edition/9781838821654 and given in this section from the same book. In **Section B** the code is provided. The output of this code in provided in **Section. C**. The code is provided in **Advanced Deep Learning with TensorFlow 2 and Keras - Second Edition**," *Packt*. [Online]. Available: https://www.packtpub.com/product/advanced-deep-learning-with-tensorflow-2-and-keras-second-edition/9781838821654

One can download the code from GitHub and run it on any Python environment. It is also explained in detail in the author book *Advanced Deep Learning with TensorFlow 2 and Keras - Second Edition*.

Supplementary Files

The *MNIST database of handwritten digits* was provided by **LeCun, Yann** http://yann.lecun.com/exdb/mnist/ once can download the MNSIT from the website above and can run it with the project code as this dataset is the main dataset for this project.

The code repository for the project is a part for the Book of *Advanced Deep Learning with TensoFlow 2. and Keras*, published by *Packt*. It contains all the supporting project files necessary to work through the project as well as the author's book. The files for the code are available on **Github.**

PacktPublishing/Advanced-Deep-Learning-with-Keras: Advanced Deep Learning with Keras, published by Packt (github.com)

Packages and libraries Needed to Run the Project Code:

1. TensorFlow 2 and above.
2. Anaconda (Highly Recommended)
3. PyCharm (Recommended)
4. numpy==1.14.5
5. numpydoc==0.8.0
6. scikit-image==0.13.1
7. scikit-learn==0.19.1
8. scipy==1.1.0
9. tensorboard==1.8.0
10. tensorflow==1.8.0
11. Keras==2.2.0
12. Keras-Applications==1.0.2
13. Keras-Preprocessing==1.0.1
14. seaborn==0.8.1

SECTION A.

Installation

Please download Anaconda from: Anaconda. To install anaconda:

```
sh <name-of-downloaded-Anaconda3-installer>
```

A machine with at least 1 NVIDIA GPU (1060 or better) is required. Follow the guidance on PacktPublishing/Advanced-Deep-Learning-with-Keras: Advanced Deep Learning with Keras, published by Packt (github.com) to install NVIDIA driver and CuDNN to enable GPU support.

The Requirements Installations:

Please Install the following on Anaconda (Highly recommended) You can Use `pip install` command to install the requirements packages or any other command.

```
numpy==1.14.5
numpydoc==0.8.0
pandas==0.23.0
scikit-image==0.13.1
scikit-learn==0.19.1
scipy==1.1.0
tensorboard==1.8.0
tensorflow==1.8.0
Keras==2.2.0
Keras-Applications==1.0.2
Keras-Preprocessing==1.0.1
seaborn==0.8.1
```

Software Install Guidance:

Here what you need to do if you never use **python** or any of the requirement's installations flowing the instructions which was provided by Dr. Ball, John at the ECE Department in Mississippi State University, Digital Signal Processing, Software Install Guide – v1.0.

If you don't have python first step you must download it and install it (Python 3.7) or above.

1. Install Anaconda

To install Anaconda, go to https://docs.anaconda.com/anaconda/install/windows/ and follow the install directions.

Note: Install to default directory. **Do not put spaces in path.**

Open Anaconda.

Update anaconda by typing:

```
conda update conda
conda update --all
```

Note: Press 'y' if prompted.

2. Configure TensorFlow-CPU in Anaconda

Open the Anaconda prompt

```
conda create -n tf python=3.7
```

Press 'y' when prompted. (Process can take a few minutes)
Note: you can activate / deactivate this environment in Anaconda by typing

```
conda activate tf
conda deactivate
```

Activate the environment.

```
conda activate tf
```

Install packages you need.

```
conda install -c conda-forge matplotlib
conda install -c anaconda numpy
conda install -c anaconda tensorflow=2.1
conda install -c conda-forge pydotplus
conda install -c conda-forge python-graphviz
```

and

```
numpydoc==0.8.0
pandas==0.23.0
scipy==1.1.0
tensorboard==1.8.0
Keras==2.2.0
Keras-Applications==1.0.2
Keras-Preprocessing==1.0.1
seaborn==0.8.1
```

3. Install PyCharm IDE

To install PyCharm, go to https://www.jetbrains.com/pycharm/download/#section=windows Make sure Windows is selected under Download PyCharm > Click Download under community > Follow directions.

4. Set Interpreter in PyCharm

Start PyCharm
If a project is loaded (you can see some code), click File > Close Project
Click Configure (at bottom right of window)
Click Settings (at top of list)
Click Project Interpreter (about ¾ down on the left side)
At top right, click down arrow in box next to Project Interpreter.
Choose the one with Python 3.7 (tf)
Click OK
If this does not show up, click the wheel, and select Add
Select Existing Environment radiobutton
Click square next to Make it available to all projects.
Choose directory for Anaconda with tf
Click OK
Repeat step above
Click OK

5. Restart the PC

Restart the PC.

SECTION. B

THE PROJECT CODE

The project code was written using Python 3 language. There are some other packages, libraries, and API that were used in this code. the code was run on python environment using Colaboratory Notebook (Colab), an interactive environment provided by Google. The output of this code in provided in Section. B. The code is provided in Advanced Deep Learning with TensorFlow 2 and Keras - Second Edition," *Packt*. [Online]. Available: https://www.packtpub.com/product/advanced-deep-learning-with-tensorflow-2-and-keras-second-edition/9781838821654

Once can download the code from GitHub and run it on any Python environment. It is also explained in detail in the author book Advanced Deep Learning with TensorFlow 2 and Keras - Second Edition

The MNIST database of handwritten digits was provided by LeCun, Yann http://yann.lecun.com/exdb/mnist/ once can download the MNSIT from the website above and can run it with the project code as this dataset is the main dataset for this project.

Google Colaboratory

```
"""
    Pattern Recognition Dr. Tang

    Adapted from Advanced Deep Learning with TensorFlow 2 and Keras by Rowel Atienza
https://github.com/PacktPublishing/Advanced-Deep-Learning-with-Keras

    John Ball, Digital Signal Processing
    13 Aug 2020 [1]
Network I
"""

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import tensorflow
import numpy as np
import tensorflow.keras.backend as K

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout, Conv2D, MaxPooling2D, Flatten
from tensorflow.keras.utils import to_categorical, plot_model
from tensorflow.keras.datasets import mnist
from confusion_matrix import confusion_matrix, print_confusion_matrix_stats


# Force code to run on the CPU
tensorflow.config.set_visible_devices([], 'GPU')

with tensorflow.device('/cpu:0'):

    # Load mnist dataset
```

```
        print("\nLoading MNIST dataset.")
        (x_train, y_train), (x_test, y_test) = mnist.load_data()

        # Compute the number of labels
        num_labels = len(np.unique(y_train))

        # Convert to one-hot vectors
        y_train = to_categorical(y_train, num_labels)
        y_test = to_categorical(y_test, num_labels)

        # Get the image dimensions (assumed square)
        image_size = x_train.shape[1]
        input_size = image_size * image_size

        # Resize and normalize
        x_train = np.reshape(x_train, [-1, image_size, image_size, 1])
        x_test = np.reshape(x_test, [-1, image_size, image_size, 1])
        x_train = x_train.astype('float32') / 255
        x_test = x_test.astype('float32') / 255

        # Network parameters
        # image is processed as is (square grayscale)
        input_shape = (image_size, image_size, 1)
        batch_size = 128
        kernel_size = 3
        pool_size = 2
        filters = 16
        dropout = 0.2

        # model is a stack of CNN-ReLU-MaxPooling
        model = Sequential()
        model.add(Conv2D(filters=filters,
                         kernel_size=kernel_size,
                         activation='relu',
                         input_shape=input_shape))
        model.add(MaxPooling2D(pool_size))
        model.add(Conv2D(filters=filters,
                         kernel_size=kernel_size,
                         activation='relu'))
        model.add(MaxPooling2D(pool_size))
        model.add(Conv2D(filters=filters,
                         kernel_size=kernel_size,
                         activation='relu'))
        model.add(Flatten())

        # dropout added as regularizer
        model.add(Dropout(dropout))

        # output layer is 10-dim one-hot vector
        model.add(Dense(num_labels))
        model.add(Activation('softmax'))

        # Print and plot model
        model.summary()
        plot_model(model, to_file='mp1_nn1.png', show_shapes=True)

        # loss function for one-hot vector
        # use of adam optimizer
        # accuracy is good metric for classification tasks
        model.compile(loss='categorical_crossentropy',
                      optimizer='adam',
                      metrics=['accuracy'])

        # train the network
        model.fit(x_train, y_train, epochs=2, batch_size=batch_size)

        # Compute predictions (test mode) for training data
        y_pred = model.predict(x_train,
                               batch_size=batch_size,
                               verbose=0)

        # Convert one-hot tensors back to class labels (and make them numpy arrays)
        y_train_true_class = K.argmax(y_train).numpy()
        y_train_pred_class = K.argmax(y_pred).numpy()

        # create CM and print confusion matrix and stats
        cm_train, cm_train_acc = confusion_matrix(y_train_true_class, y_train_pred_class)
        print_confusion_matrix_stats(cm_train, 'Train')

        # Validate the model on test dataset to determine generalization
        y_pred = model.predict(x_test,
                               batch_size=batch_size,
                               verbose=0)

        # Convert one-hot tensors back to class labels (and make them numpy arrays)
        y_test_true_class = K.argmax(y_test).numpy()
        y_test_pred_class = K.argmax(y_pred).numpy()

        # create CM and print confusion matrix and stats
        cm_test, cm_test_acc = confusion_matrix(y_train_true_class, y_train_pred_class)
        print_confusion_matrix_stats(cm_test, 'Test')

        # Network II
        from __future__ import absolute_import
        from __future__ import division
        from __future__ import print_function

        import tensorflow
        import numpy as np
        import tensorflow.keras.backend as K

        import matplotlib.pyplot as plt
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense, Activation, Dropout
        from tensorflow.keras.utils import to_categorical, plot_model
        from tensorflow.keras.datasets import mnist
        from confusion_matrix import confusion_matrix, print_confusion_matrix_stats

        # Force code to run on the CPU
        tensorflow.config.set_visible_devices([], 'GPU')

        with tensorflow.device('/cpu:0'):
            # Load mnist dataset
            print("\nLoading MNIST dataset.")
```

```
        (x_train, y_train), (x_test, y_test) = mnist.load_data()

        # Compute the number of labels
        num_labels = len(np.unique(y_train))

        # Convert to one-hot vectors
        y_train = to_categorical(y_train)
        y_test = to_categorical(y_test)

        # Get the image dimensions (assumed square)
        image_size = x_train.shape[1]
        input_size = image_size * image_size

        # Resize and normalize
        x_train = np.reshape(x_train, [-1, input_size])
        x_train = x_train.astype('float32') / 255
        x_test = np.reshape(x_test, [-1, input_size])
        x_test = x_test.astype('float32') / 255

        # Setup the network parameters
        batch_size = 128
        hidden_units = 32
        dropout = 0.45

        # model is a 3-layer MLP with ReLU and dropout after each layer
        model = Sequential()
        model.add(Dense(hidden_units, input_dim=input_size))
        model.add(Activation('relu'))
        model.add(Dropout(dropout))
        model.add(Dense(hidden_units))
        model.add(Activation('relu'))
        model.add(Dropout(dropout))
        model.add(Dense(num_labels))

        # this is the output for one-hot vector
        model.add(Activation('softmax'))

        # Print model summary and save the network image to the file specified
        model.summary()
        plot_model(model, to_file='mp1_nn1.png', show_shapes=True)

        # loss function for one-hot vector
        # use of adam optimizer
        # accuracy is good metric for classification tasks
        model.compile(loss='categorical_crossentropy',
                      optimizer='adam',
                      metrics=['accuracy'])

        # Train the network
        model.fit(x_train, y_train, epochs=20, batch_size=batch_size)

        # Compute predictions (test mode) for training data
        y_pred = model.predict(x_train,
                               batch_size=batch_size,
                               verbose=0)

        # Convert one-hot tensors back to class labels (and make them numpy arrays)
        y_train_true_class = K.argmax(y_train).numpy()
        y_train_pred_class = K.argmax(y_pred).numpy()

        # create CM and print confusion matrix and stats
        cm_train, cm_train_acc = confusion_matrix(y_train_true_class, y_train_pred_class)
        print_confusion_matrix_stats(cm_train, 'Train')

        # Validate the model on test dataset to determine generalization
        y_pred = model.predict(x_test,
                               batch_size=batch_size,
                               verbose=0)

        # Convert one-hot tensors back to class labels (and make them numpy arrays)
        y_test_true_class = K.argmax(y_test).numpy()
        y_test_pred_class = K.argmax(y_pred).numpy()

        # create CM and print confusion matrix and stats
        cm_test, cm_test_acc = confusion_matrix(y_test_true_class, y_test_pred_class)
        print_confusion_matrix_stats(cm_test, 'Test')

    # Modefied Network

    from __future__ import absolute_import
    from __future__ import division
    from __future__ import print_function

    import tensorflow
    import numpy as np
    import tensorflow.keras.backend as K

    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense, Activation, Dropout, Conv2D, MaxPooling2D, Flatten
    from tensorflow.keras.utils import to_categorical, plot_model
    from tensorflow.keras.datasets import mnist
    from confusion_matrix import confusion_matrix, print_confusion_matrix_stats


    # Force code to run on the CPU
    tensorflow.config.set_visible_devices([], 'GPU')

    with tensorflow.device('/cpu:0'):

        # Load mnist dataset
        print("\nLoading MNIST dataset.")
        (x_train, y_train), (x_test, y_test) = mnist.load_data()

        # Compute the number of labels
        num_labels = len(np.unique(y_train))

        # Convert to one-hot vectors
        y_train = to_categorical(y_train, num_labels)
        y_test = to_categorical(y_test, num_labels)

```

```
        # Get the image dimensions (assumed square)
        image_size = x_train.shape[1]
        input_size = image_size * image_size

        # Resize and normalize
        x_train = np.reshape(x_train, [-1, image_size, image_size, 1])
        x_test = np.reshape(x_test, [-1, image_size, image_size, 1])
        x_train = x_train.astype('float32') / 255
        x_test = x_test.astype('float32') / 255

        # Network parameters
        # image is processed as is (square grayscale)
        input_shape = (image_size, image_size, 1)
        batch_size = 64
        kernel_size = 3
        pool_size = 2
        filters = 64
        dropout = 0.3

        # model is a stack of CNN-ReLU-MaxPooling
        model = Sequential()
        model.add(Conv2D(filters=filters,
                         kernel_size=kernel_size,
                         activation='relu',
                         input_shape=input_shape))
        model.add(MaxPooling2D(pool_size))
        model.add(Conv2D(filters=filters,
                         kernel_size=kernel_size,
                         activation='relu'))
        model.add(MaxPooling2D(pool_size))
        model.add(Conv2D(filters=filters,
                         kernel_size=kernel_size,
                         activation='relu'))
        model.add(Flatten())

        # dropout added as regularizer
        model.add(Dropout(dropout))

        # output layer is 10-dim one-hot vector
        model.add(Dense(num_labels))
        model.add(Activation('softmax'))

        # Print and plot model
        model.summary()
        plot_model(model, to_file='mp1_nn1.png', show_shapes=True)

        # loss function for one-hot vector
        # use of adam optimizer
        # accuracy is good metric for classification tasks
        model.compile(loss='categorical_crossentropy',
                      optimizer='adam',
                      metrics=['accuracy'])

        # train the network
        model.fit(x_train, y_train, epochs=2, batch_size=batch_size)

        # Compute predictions (test mode) for training data
        y_pred = model.predict(x_train,
                               batch_size=batch_size,
                               verbose=0)

        # Convert one-hot tensors back to class labels (and make them numpy arrays)
        y_train_true_class = K.argmax(y_train).numpy()
        y_train_pred_class = K.argmax(y_pred).numpy()

        # Students, insert code here to create CM and print confusion matrix and stats
        cm_train, cm_train_acc = confusion_matrix(y_train_true_class, y_train_pred_class)
        print_confusion_matrix_stats(cm_train, 'Train')

        # Validate the model on test dataset to determine generalization
        y_pred = model.predict(x_test,
                               batch_size=batch_size,
                               verbose=0)

        # Convert one-hot tensors back to class labels (and make them numpy arrays)
        y_test_true_class = K.argmax(y_test).numpy()
        y_test_pred_class = K.argmax(y_pred).numpy()

        # Students, insert code here to create CM and print confusion matrix and stats
        cm_test, cm_test_acc = confusion_matrix(y_train_true_class, y_train_pred_class)
        print_confusion_matrix_stats(cm_test, 'Test')
```

SECTION C.
The output of the Code

This is the output of the code shown in Section A. the code was run on python environment using Colaboratory Notebook (Colab), an interactive environment provided by Google,

```
THE OUTPUT OF NETWORK I
Loading MNIST dataset.
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 26, 26, 16)        160       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 16)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 16)        2320      
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 16)          0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 16)          2320      
_________________________________________________________________
flatten (Flatten)            (None, 144)               0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 144)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1450      
_________________________________________________________________
activation_3 (Activation)    (None, 10)                0         
=================================================================
Total params: 6,250
Trainable params: 6,250
Non-trainable params: 0
_________________________________________________________________
Epoch 1/2
469/469 [==============================] - 21s 44ms/step - loss: 1.0608 - accuracy: 0.6598
Epoch 2/2
469/469 [==============================] - 20s 44ms/step - loss: 0.1790 - accuracy: 0.9436

Train Confusion Matrix: (Rows are true classes, columns predictions)

     5813         1        14         2         2        21        35         2        24         9
        0      6641        36        15        12         4         2        13        19         0
        3        20      5796        42         9         3         8        31        41         5
        2         4        61      5959         0        40         1        23        25        16
        3         6         5         3      5627         0        27         8        27       136
        8        13         4        57         4      5271        18         6        33         7
       15         6         3         0        14        26      5833         0        21         0
        6        19        79        20        21         8         0      6050        10        52
        6        12        20        41        12        30        15         7      5674        34
       23         8         1        43        38        59         3        60        46      5668


Confusion matrix Train stats:
```

```
Total instances          : 60000
Overall accuracy    (OA)  : 0.97220


Test Confusion Matrix: (Rows are true classes, columns predictions)

     5813         1        14         2         2        21        35         2        24         9
        0      6641        36        15        12         4         2        13        19         0
        3        20      5796        42         9         3         8        31        41         5
        2         4        61      5959         0        40         1        23        25        16
        3         6         5         3      5627         0        27         8        27       136
        8        13         4        57         4      5271        18         6        33         7
       15         6         3         0        14        26      5833         0        21         0
        6        19        79        20        21         8         0      6050        10        52
        6        12        20        41        12        30        15         7      5674        34
       23         8         1        43        38        59         3        60        46      5668

Confusion matrix Test stats:
Total instances          : 60000
Overall accuracy    (OA)  : 0.97220

OUTPUT NETWORK II
Loading MNIST dataset.
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 32)                25120
_________________________________________________________________
activation (Activation)      (None, 32)                0
_________________________________________________________________
dropout (Dropout)            (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                1056
_________________________________________________________________
activation_1 (Activation)    (None, 32)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                330
_________________________________________________________________
activation_2 (Activation)    (None, 10)                0
=================================================================
Total params: 26,506
Trainable params: 26,506
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
469/469 [==============================] - 2s 2ms/step - loss: 1.5870 - accuracy: 0.4418
Epoch 2/20
469/469 [==============================] - 1s 2ms/step - loss: 0.7469 - accuracy: 0.7626
Epoch 3/20
469/469 [==============================] - 1s 2ms/step - loss: 0.6355 - accuracy: 0.8034
Epoch 4/20
469/469 [==============================] - 1s 2ms/step - loss: 0.5786 - accuracy: 0.8203
Epoch 5/20
469/469 [==============================] - 1s 2ms/step - loss: 0.5289 - accuracy: 0.8370
Epoch 6/20
469/469 [==============================] - 1s 2ms/step - loss: 0.5073 - accuracy: 0.8458
Epoch 7/20
469/469 [==============================] - 1s 2ms/step - loss: 0.4939 - accuracy: 0.8497
Epoch 8/20
469/469 [==============================] - 1s 2ms/step - loss: 0.4817 - accuracy: 0.8515
Epoch 9/20
469/469 [==============================] - 1s 2ms/step - loss: 0.4764 - accuracy: 0.8533
Epoch 10/20
469/469 [==============================] - 1s 2ms/step - loss: 0.4607 - accuracy: 0.8565
Epoch 11/20
469/469 [==============================] - 1s 2ms/step - loss: 0.4634 - accuracy: 0.8584
Epoch 12/20
469/469 [==============================] - 1s 2ms/step - loss: 0.4540 - accuracy: 0.8610
Epoch 13/20
469/469 [==============================] - 1s 2ms/step - loss: 0.4554 - accuracy: 0.8597
Epoch 14/20
```

```
469/469 [==============================] - 1s 2ms/step - loss: 0.4490 - accuracy: 0.8605
Epoch 15/20
469/469 [==============================] - 1s 2ms/step - loss: 0.4311 - accuracy: 0.8667
Epoch 16/20
469/469 [==============================] - 1s 2ms/step - loss: 0.4365 - accuracy: 0.8650
Epoch 17/20
469/469 [==============================] - 1s 2ms/step - loss: 0.4274 - accuracy: 0.8643
Epoch 18/20
469/469 [==============================] - 1s 2ms/step - loss: 0.4313 - accuracy: 0.8657
Epoch 19/20
469/469 [==============================] - 1s 2ms/step - loss: 0.4268 - accuracy: 0.8659
Epoch 20/20
469/469 [==============================] - 1s 2ms/step - loss: 0.4219 - accuracy: 0.8679


Train Confusion Matrix: (Rows are true classes, columns predictions)

     5855         0         7         0         6         3        16         1        34         1
        2      6579        36        14         3         3         4        19        73         9
       29        23      5720        37        34         3         8        33        69         2
       12        11       127      5750         2       118         4        44        42        21
       15        19        24         1      5569         3        39         3        21       148
       41         3        18        76        17      5054        65         0       107        40
       47        10         4         0        20        39      5777         0        21         0
       19        23        62        19        34         1         1      6032         7        67
       41        55        45        38        18        51        31         6      5535        31
       32         7         0        21        92        79         3        63        48      5604

Confusion matrix Train stats:
Total instances          : 60000
Overall accuracy    (OA)  : 0.95792


Test Confusion Matrix: (Rows are true classes, columns predictions)

      970         0         1         1         1         1         2         2         2         0
        0      1112         3         2         0         0         3         1        13         1
        6         2       986         9         5         1         3         6        13         1
        1         2        19       949         1        20         0         9         6         3
        0         0         5         1       923         0        16         2         6        29
       10         1         2        21         5       813        10         1        19        10
       10         5         2         0         4         9       926         0         2         0
        2         7        22         6         4         0         0       970         2        15
        8         6         6         9         6        12         6         8       912         1
        7         5         1         4        17        19         0         8         6       942

Confusion matrix Test stats:
Total instances          : 10000
Overall accuracy    (OA)  : 0.95030


THE OUTPUT OF MODEFIED NETWORK
Loading MNIST dataset.
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 64)        640
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 576)               0
_________________________________________________________________
dropout (Dropout)            (None, 576)               0
_________________________________________________________________
dense (Dense)                (None, 10)                5770
```

```
_________________________________________________________________
activation (Activation)      (None, 10)                0
=================================================================
Total params: 80,266
Trainable params: 80,266
Non-trainable params: 0
_________________________________________________________________
Epoch 1/2
938/938 [==============================] - 83s 87ms/step - loss: 0.4622 - accuracy: 0.8550
Epoch 2/2
938/938 [==============================] - 82s 87ms/step - loss: 0.0660 - accuracy: 0.9802
```

Train Confusion Matrix: (Rows are true classes, columns predictions)

5903	0	0	2	2	1	7	0	4	4
1	6684	19	0	7	1	3	13	12	2
7	6	5902	7	6	0	3	14	10	3
1	0	26	6065	0	15	1	6	11	6
2	3	0	0	5812	0	8	1	6	10
7	1	3	12	2	5335	34	0	22	5
8	0	2	0	4	2	5892	0	10	0
2	9	29	9	12	1	0	6183	6	14
8	1	10	2	9	5	8	3	5789	16
13	4	1	5	35	9	3	24	10	5845

```
Confusion matrix Train stats:
Total instances          : 60000
Overall accuracy    (OA)  : 0.99017
```

Test Confusion Matrix: (Rows are true classes, columns predictions)

5903	0	0	2	2	1	7	0	4	4
1	6684	19	0	7	1	3	13	12	2
7	6	5902	7	6	0	3	14	10	3
1	0	26	6065	0	15	1	6	11	6
2	3	0	0	5812	0	8	1	6	10
7	1	3	12	2	5335	34	0	22	5
8	0	2	0	4	2	5892	0	10	0
2	9	29	9	12	1	0	6183	6	14
8	1	10	2	9	5	8	3	5789	16
13	4	1	5	35	9	3	24	10	5845

```
Confusion matrix Test stats:
Total instances          : 60000
Overall accuracy    (OA)  : 0.99017
```

Google Colaboratory