Lecture Notes Lec5-1-mnist

MNIST Convolutional Neural Network

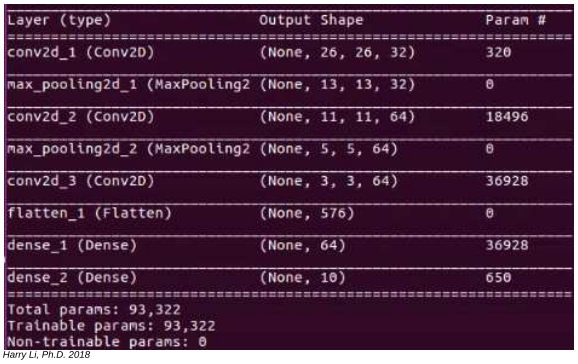
Harry Li ‡, Ph.D.
Computer Engineering Department, San Jose State University
San Jose, CA 95192, USA

Email†: harry.li@ctione.com

***Abstract*—This note describes the techniques MNIST Convolutional Neural Networks (CNN) for hand written digits recognition.**

I. Architecture of MNIST

The MNIST archiecture consists of $C_1 M_1 \ C_2 M_2 \ C_3 \ F_{576} D_{64} D_{10}$, and it is illustrated below.



Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_1 (MaxPooling2	(None, 13, 13, 32)	0
conv2d_2 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_2 (MaxPooling2	(None, 5, 5, 64)	0
conv2d_3 (Conv2D)	(None, 3, 3, 64)	36928
flatten_1 (Flatten)	(None, 576)	0
dense_1 (Dense)	(None, 64)	36928
dense_2 (Dense)	(None, 10)	650

Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0

Harry Li, Ph.D. 2018

Fig. 1. The MNIST archiecture consists of $C_1 M_1 C_2 M_2 C_3 F_{576} D_{64} D_{10}$.

The graphical illustration of the architecture is given in the second figure. Note this graphical representation is little bit different than the one we have actually implemented in this note.

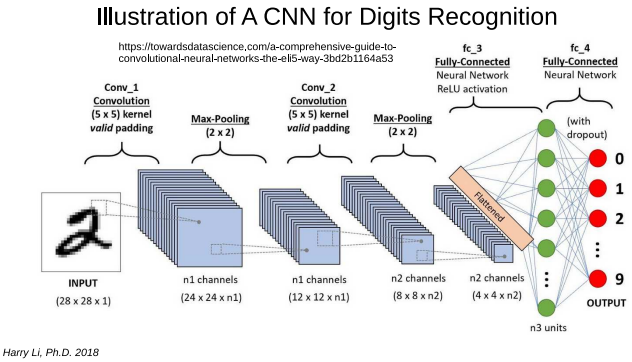


Fig. 2. The MNIST archiecture consists of $C_1 M_1 C_2 M_2 C_3 F_{576} D_{64} D_{10}$.

To test run the code, open terminal, then
$python3 7-1convnets-NumeralDet-saveTrained.py

II. Python Code to Build MNIST CNN

We list the full listing of the python program 7-1convnets-NumeralDet-saveTrained.py below.

```
#-------------------------------------------------------------*
# program : 7-1convnets-NumeralDetection-ch05.py;            *
# date    : Oct 18, 2018                                     *
# version : x0.10;                    status: tested;        *
# ref: https://github.com/fchollet/                          *
#      deep-learning-with-python-notebooks/blob/             *
#      96d58b5727fcf76106f929f5ce24c40fc9b46d75              *
#                                                            *
# purpose : demo of saving trained mnist net                 *
#-------------------------------------------------------------*
import keras
keras.__version__

#--------------build convnet--------------------*
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.summary() #check the model

#-----------flatten then 10-way classifier--------*
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

#-----------get NIST image data--------------------*
from keras.datasets import mnist
from keras.utils import to_categorical

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

#-------------train----------------------*
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=64)

test_loss, test_acc = model.evaluate(test_images, test_labels)
test_acc

#-------------save trained model---------*
import h5py
model.save('harryTest.h5')
#-end
```

A. Pseudo Code of the MINST Architecture

B. Code of Convolutional Layers

Note in the source code implementation given in the section above, the convolutional C_1 layer is given by the TF Keras function

```
model.add(layers.Conv2D(32,(3,3),
 activation='relu',input\_shape=(28,28,1)))
```

From the code, we know

1. The number of convolutions to be carried out is 32;

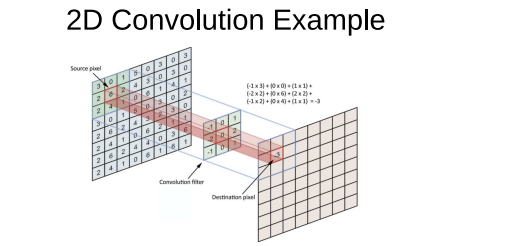
Algorithm 1 Build MNIST convolutional NN

Require: import keras
from keras import layers
from keras import model

Ensure: Build MNIST convolutional NN layers sequentially
model = models.Sequential()
model.add(layers.Conv2D(32,(3,3), activation='relu', input_shape=(28,28,1)))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64,(3,3),activation='relu'))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64,(3,3),activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64,activation='relu'))
model.add(layers.Dense(10,activation='softmax'))

2. The size of the convolution kernel is 3x3;
3. The activation function $f(W \cdot X)$ is relu; and
4. The input shape is 28x28 and it is one single layer.

Note the dimension change from C_1M_2 to C_2M_2 is shown in the above section, see Figure 1 of the model.summary() output, we have the dimension change to 26x26, and 32 which indicated total of 32 planes due to 32 convolutions. The following figure gives the general description of 2D convolution.



https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac

Harry Li, Ph.D. 2016, 2018, 2020

Fig. 3. Illustration of 2D convolution.

C. Code of Flatten Layer

Note in the source code implementation given in the section below, the flatten layer is given by the TF Keras function

```
model.add(layers.Flatten())
```

which is placed right after the convolutional layer C_3. From the code

```
model.add(layers.Conv2D(64,(3,3),activation='relu'))
```

We know the size of each plane is 3x3, and there are total 64 layers, as shown below. Therefore, we have 576 neurons (576 = 3x3x64).

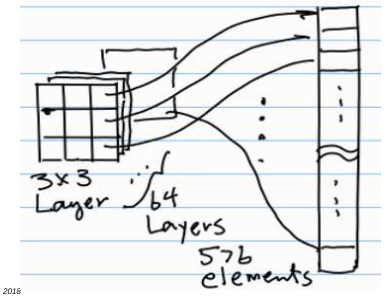


Fig. 4. The flatten layer from the convolutional layer C_3.

TABLE I
TF-KERAS FUNCTION TABLE 1

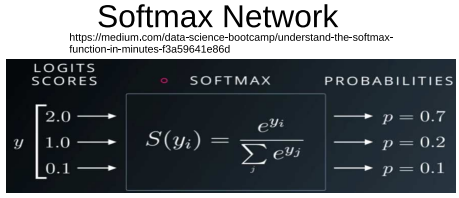
Syntax Description	Note
`model.add(layers.Flatten())`	model
`layers.Flatten()`	flatten L
`layers.Conv2D(64,(3,3),activation='relu'))`	2D Conv

III. SOFTMAX ACTIVATION FUNCTION

The last layer (output) of MNIST uses softmax activation function as below,

```
model.add(layers.Dense(10,activation='softmax'))
```

To understand this softmax activation function, let's take a look at the following example.



The above Udacity lecture slide shows that Softmax function turns logits [2.0, 1.0, 0.1] into probabilities [0.7, 0.2, 0.1], and the probabilities sum to 1.

"logits layer is popularly used for the last neuron layer for classification task which produces raw prediction values as real numbers ranging from [-infinity, +infinity]"

"Softmax's input is the output of the fully connected layer immediately preceeding it, and it outputs the final output of the entire neural network. This output is a probability distribution of all the label class candidates."

Harry Li, Ph.D. 2018

Fig. 5. Illustration of softmax function.

The above example is originally from Udacity site example which shows a simple calculation of a Softmax function. IN this example, three output digits [2.0, 1.0, 0.1] from a NN into their corrensponding probabilities [0.7, 0.2, 0.1]. We know the sum of their probabilities has to be equal to 1. Following the formula in the figure, we can form softmax function for the number N outputs. Try it, for example, for [110, -35.1, 0.2, 1.1, 10], use softmax to find their corresponding probalities.

The softmax function is defined as follows,

$$f(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{1}$$

This activation function is popularly used for the last neuron layer for classification task which produces raw prediction values as real numbers ranging from [-infinity, +infinity]. Then with activation function of softmax, we can change them to probabilities. Softmaxs input is the output of the fully connected layer, and it is the final output of the entire neural network. This output is a probability distribution of all the label class candidates.

IV. Relu Activation Function

The convolutional C_1 layer uses activation function relu (Rectified Linear Unit),

```
model.add(layers.Conv2D(32,(3,3),
 activation='relu',input\_shape=(28,28,1)))
```

whose plot is shown in the following figure. We need non-linear activation function to make NN to be able to perform classification tasks which can only be separated by nonlinear decision making functions. ReLU is the simplest non-linear function and its efficient for computation.

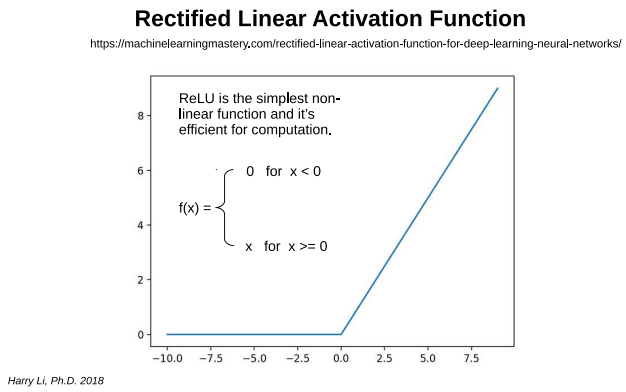


Fig. 6. Illustration of Relu function.

V. Python Code to Train MNIST CNN

A. Python Code to Load Dataset for Training MNIST CNN

```
#-----------get NIST image data-------------------*
from keras.datasets import mnist
from keras.utils import to_categorical

(train_images,train_labels),(test_images,test_labels)
    =mnist.load_data()

train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

B. Python Code to Train MNIST CNN

```
#-------------train----------------------*
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5,
      batch_size=64)

test_loss, test_acc = model.evaluate(test_images,
      test_labels)
test_acc
```

To train the MNIST we use model.compile which configures the model for training [TensorFlow].

```
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Note:

1. The optimizer is "rmsprop". RMSprop is a gradient based optimization technique used in training neural networks. This technique balances the step size, e.g., momentum, which makes the step size decrease for large gradients to avoid exploding, and increase the step for small gradients to avoid vanishing. So here is the background of it. *"RMSprop is unpublished optimization algorithm designed for neural networks, first proposed by Geoff Hinton in lecture 6 of the online course Neural Networks for Machine Learning . RMSprop lies in the realm of adaptive learning rate methods, which have been growing in popularity in recent years, but also getting some criticism[6]. Its famous for not being published, yet being very well-known; most deep learning framework include the implementation of it out of the box"* [Bushaev, 2018].
2. The loss function or the objective function is "categorical_crossentropy", which minimzes the loss function based on the probability of the likelihood of the predicted output class.
3. The metrics is "['accuracy']".

C. Python Code to Save Trained MNIST CNN

```
#-------------save trained model---------*
import h5py
model.save('harryTest.h5')
#-end
```


Acknowledgment

I would like to express my thanks to CTI One Deep Learning Team for their coding and participation in the deep learning classes which lead to the enhancement of AIV100 project.

References

[1] [Liu, 2017] Danqing Liu, Relu Practical Guide, https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7

[2] [Tensor Flow, 2021] https://www.tensorflow.org/api_docs /python/tf/keras/Model

[3] [Bushaev, 2018], Vitaly Bushaev, Understanding RMSprop, faster neural network learning, https://towardsdatascience.com/understanding-rmsprop-faster-neural- network-learning-62e116fcf29a