

School of Electrical and Electronic Engineering
Singapore Polytechnic
Machine Learning and AI (ET0732)

Topic FOUR

Computer Vision

Lab 4_1: Classify images with a Linear model

In this hands-on lab of this section, you'll learn how to classify images of flowers with a linear model using the tf.keras API.

Lab Link: [Classifying Images with a Linear Model | Google Cloud Skills Boost](#) (5 credits)

Lab Instructions:

1. In the notebook interface, navigate to **training-data-analyst > courses > machine_learning > deepdive2 > computer_vision_fun > labs** and open **classifying_images_with_a_linear_model.ipynb**.
2. In the notebook interface, click **Edit > Clear All Outputs**.
3. Carefully read through the notebook instructions and **fill in lines marked with #TODO** where you need to complete the code.

Tip: To run the current cell, click the cell and press SHIFT+ENTER. Other cell commands are listed in the notebook UI under **Run**.

- Hints may also be provided for the tasks to guide you. Highlight the text to read the hints, which are in white text.
- To view the complete solution, navigate to **training-data-analyst > courses > machine_learning > deepdive2 > computer_vision_fun > solutions**, and open **classifying_images_with_a_linear_model.ipynb**.

Lab 4_2: Classifying Images with a NN and DNN Model

In this lab, you'll learn how to build a neural network and a Deep Neural Network Model to classify the tf-flowers dataset.

Lab Link: [Classifying Images with a NN and DNN Model | Google Cloud Skills Boost](#) (5 credits)

Lab Instructions:

1. In the notebook interface, navigate to **training-data-analyst > courses > machine_learning > deepdive2 > computer_vision_fun > labs** and open **classifying_images_with_a_nn_and_dnn_model.ipynb**.
2. In the notebook interface, click **Edit > Clear All Outputs**.
3. Carefully read through the notebook instructions and **fill in lines marked with #TODO** where you need to complete the code.

Tip: To run the current cell, click the cell and press SHIFT+ENTER.

- Hints may also be provided for the tasks to guide you. Highlight the text to read the hints, which are in white text.

School of Electrical and Electronic Engineering
Singapore Polytechnic
Machine Learning and AI (ET0732)

- To view the complete solution, navigate to **training-data-analyst > courses > machine_learning > deeplive2 > computer_vision_fun > solutions**, and open **classifying_images_with_a_nn_and_dnn_model.ipynb**.
-

Lab 4_3: Handwritten Digit Recognition using DNN & CNN

Introduction

Handwritten digit recognition is a common image recognition task where computers recognize digits in handwritten images. Different from printed fonts, handwriting of different people have different styles and sizes, making it difficult for computers to recognize handwriting. This exercise uses deep learning and TensorFlow tools to train a model using MNIST datasets.

This section describes the basic process of TensorFlow computing and basic elements for building DNN and CNN network, including dataset acquisition, network model building, model training, and model validation.

Tasks

- Read the MNIST handwritten digit dataset.
- Get started with TensorFlow by using simple mathematical models.
- Implement softmax regression by using high-level APIs.
- Build a multi-layer CNN.
- Implement a CNN by using high-level APIs.
- Visualize prediction results.

Step 1: Data Acquisition and Processing

a. About the Dataset

- The MNIST dataset is provided by the National Institute of Standards and Technology (NIST).
- It consists of handwritten digits from 250 different individuals, of which 50% are high school students and 50% are staff from Bureau of the Census.
- You can download the dataset from <http://yann.lecun.com/exdb/mnist/>, which consists of the following parts:
 - Training set images: train-images-idx3-ubyte.gz (9.9 MB, 47 MB after decompression, including 60,000 samples)
 - Training set labels: train-labels-idx1-ubyte.gz (29 KB, 60 KB after decompression, including 60,000 labels)
 - Test set images: t10k-images-idx3-ubyte.gz (1.6 MB, 7.8 MB after decompression, including 10,000 samples)
 - Test set labels: t10k-labels-idx1-ubyte.gz (5 KB, 10 KB after decompression, including 10,000 labels)
 - It also contains one label corresponding to each image to clarify the correct digit. For example, the labels for the following four images are 5, 0, 4, and 1.

School of Electrical and Electronic Engineering
Singapore Polytechnic
Machine Learning and AI (ET0732)



b. MNIST Dataset Reading

The dataset can be loaded from Tensorflow framework with the code as below:

```
import os
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, optimizers, datasets
from matplotlib import pyplot as plt
import numpy as np

(x_train_raw, y_train_raw), (x_test_raw, y_test_raw) = datasets.mnist.load_data()

print(y_train_raw[0])
print(x_train_raw.shape, y_train_raw.shape)
print(x_test_raw.shape, y_test_raw.shape)

# Convert the labels into one-hot codes.
num_classes = 10
y_train = keras.utils.to_categorical(y_train_raw, num_classes)
y_test = keras.utils.to_categorical(y_test_raw, num_classes)
print(y_train[0])
```

Output:

```
5
(60000, 28, 28) (60000,)
(10000, 28, 28) (10000,)
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

In the MNIST dataset, the **images** is a tensor in the shape of [60000, 28, 28]. The first dimension is used to extract images, and the second and third dimensions are used to extract pixels in each image. Each element in this tensor indicates the strength of a pixel in an image. The value ranges from **0** to **255**.

The label data is converted from scalar to one-hot vectors. In a one-hot vector, one digit is 1 and digits in other dimensions are all 0s. For example, label 1 can be represented as [0,1,0,0,0,0,0,0,0,0]. Therefore, the labels are a digital matrix of [60000, 10].

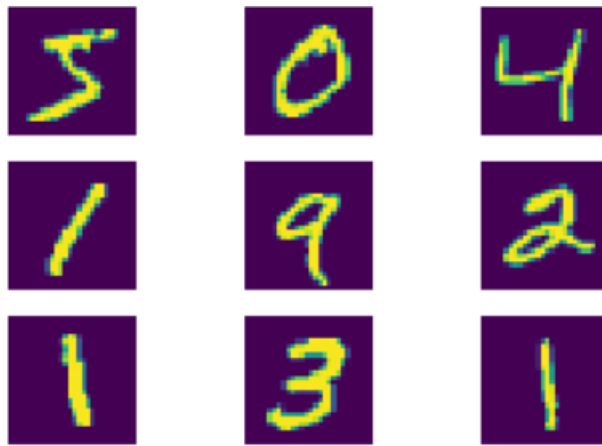
School of Electrical and Electronic Engineering
Singapore Polytechnic
Machine Learning and AI (ET0732)

c. Dataset Preprocessing and Visualization

- Data Visualization: Draw the first nine images

```
plt.figure()
for i in range(9):
    plt.subplot(3,3,i+1)
    plt.imshow(x_train_raw[i])
    #plt.ylabel(y[i].numpy())
    plt.axis('off')
plt.show()
```

Output:



- Data processing: The output of a fully connected network must be in the form of vector, instead of the matrix form of the current images. Therefore, you need to sort the images into vectors.

Code:

```
# Convert a 28 x 28 image to a 784 x 1 vector.
x_train = x_train_raw.reshape(60000, 784)
x_test = x_test_raw.reshape(10000, 784)
```

Currently, the dynamic range of pixels is 0 to 255. Image pixels are usually normalized to the range of 0 to 1 during processing of image pixel values.

Code:

```
# Normalize image pixel values.
x_train = x_train.astype('float32')/255
x_test = x_test.astype('float32')/255
```

School of Electrical and Electronic Engineering
Singapore Polytechnic
Machine Learning and AI (ET0732)

Step 2: DNN Construction

- Building a DNN Model

Code:

```
# Create a deep neural network (DNN) model that consists of three fully connected layers and two ReLU activation functions.
model = keras.Sequential([
    layers.Dense(512, activation='relu', input_dim = 784),
    layers.Dense(256, activation='relu'),
    layers.Dense(124, activation='relu'),
    layers.Dense(num_classes, activation='softmax')])

model.summary()
```

layer.Dense() indicates a fully connected layer, and **activation** indicates a used activation function.

Output:

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 512)	401,920
dense_1 (Dense)	(None, 256)	131,328
dense_2 (Dense)	(None, 124)	31,868
dense_3 (Dense)	(None, 10)	1,250

Total params: 566,366 (2.16 MB)

Trainable params: 566,366 (2.16 MB)

Non-trainable params: 0 (0.00 B)

- Compiling the DNN Model

Code:

```
Optimizer = optimizers.Adam(0.001)
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=Optimizer,
              metrics=['accuracy'])
```

In the preceding example, the loss function of the model is defined as cross entropy, and the optimization algorithm is the **Adam** gradient descent method.

- Training the DNN Model

Code:

```
# Fit the training data to the model by using the fit method.
model.fit(x_train, y_train,
          batch_size=128,
          epochs=10,
          verbose=1)
```

School of Electrical and Electronic Engineering
Singapore Polytechnic
Machine Learning and AI (ET0732)

Output:

```
Epoch 1/10
60000/60000 [=====] - 7s 114us/sample - loss: 0.2281 - acc: 0.9327s - loss: 0.2594 -
acc: 0. - ETA: 1s - loss: 0.2535 - acc: 0.9 - ETA: 1s - loss:
Epoch 2/10
60000/60000 [=====] - 8s 129us/sample - loss: 0.0830 - acc: 0.9745s - loss: 0.0814 -
ac
Epoch 3/10
60000/60000 [=====] - 8s 127us/sample - loss: 0.0553 - acc: 0.9822
Epoch 4/10
60000/60000 [=====] - 7s 117us/sample - loss: 0.0397 - acc: 0.9874s - los
Epoch 5/10
60000/60000 [=====] - 8s 129us/sample - loss: 0.0286 - acc: 0.9914
Epoch 6/10
60000/60000 [=====] - 8s 136us/sample - loss: 0.0252 - acc: 0.9919
Epoch 7/10
60000/60000 [=====] - 8s 129us/sample - loss: 0.0204 - acc: 0.9931s - lo
Epoch 8/10
60000/60000 [=====] - 8s 135us/sample - loss: 0.0194 - acc: 0.9938
Epoch 9/10
60000/60000 [=====] - 7s 109us/sample - loss: 0.0162 - acc: 0.9948
Epoch 10/10
60000/60000 [=====] - ETA: 0s - loss: 0.0149 - acc: 0.994 - 7s 117us/sample - loss:
0.0148 - acc: 0.9948
```

epoch indicates a specific round of training. In the preceding example, full data is iterated for 10 times.

– Evaluating the DNN Model

Code:

```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Output:

```
Test loss: 0.48341113169193267
Test accuracy: 0.8765
```

According to the evaluation, the model accuracy is 0.87 after 10 model training iterations.

– Saving the DNN Model

Code:

```
logdir='./mnist_model'
if not os.path.exists(logdir):
    os.mkdir(logdir) |
model.save(logdir+'final_DNN_model.h5')
```

School of Electrical and Electronic Engineering
Singapore Polytechnic
Machine Learning and AI (ET0732)

Step 3: CNN Construction

The conventional CNN construction method helps you better understand the internal network structure but requires a large code volume. Therefore, attempts to construct a CNN by using high-level APIs are made to simplify the network construction process.

– Building a CNN Model

Code:

```
import tensorflow as tf
from tensorflow import keras
import numpy as np

model=keras.Sequential() # Create a network sequence.
## Add the first convolutional layer and pooling layer.
model.add(keras.layers.Conv2D(filters=32, kernel_size = 5, strides = (1,1),
                             padding = 'same', activation = tf.nn.relu, input_shape = (28,28,1)))
model.add(keras.layers.MaxPool2D(pool_size=(2,2), strides = (2,2), padding = 'valid'))
## Add the second convolutional layer and pooling layer.
model.add(keras.layers.Conv2D(filters=64, kernel_size = 3, strides = (1,1), padding = 'same', activation = tf.nn.relu))
model.add(keras.layers.MaxPool2D(pool_size=(2,2), strides = (2,2), padding = 'valid'))
## Add a dropout layer to reduce overfitting.
model.add(keras.layers.Dropout(0.25))
model.add(keras.layers.Flatten())
## Add two fully connected layers.
model.add(keras.layers.Dense(units=128, activation = tf.nn.relu))
model.add(keras.layers.Dropout(0.5))
model.add(keras.layers.Dense(units=10, activation = tf.nn.softmax))
```

In the preceding network, two convolutional layers and pooling layers are first added by using **keras.layers**. Afterwards, a dropout layer is added to prevent overfitting. Finally, two full connection layers are added.

– Compiling and Training the CNN Model

Code:

```
# Expand data dimensions to adapt to the CNN model.
X_train=x_train.reshape(60000,28,28,1)
X_test=x_test.reshape(10000,28,28,1)
model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=['accuracy'])
model.fit(x=X_train, y=y_train, epochs=5, batch_size=128)
```

Output:

```
Epoch 1/5
55000/55000 [=====] - 49s 899us/sample - loss: 0.2107 - acc: 0.9348

Epoch 2/5
55000/55000 [=====] - 48s 877us/sample - loss: 0.0793 - acc: 0.9763
```

```
Epoch 4/5
55000/55000 [=====] - 48s 867us/sample - loss: 0.0501 - acc: 0.9846
Epoch 5/5
55000/55000 [=====] - 50s 901us/sample - loss: 0.0452 - acc: 0.9862
```

During training, the network training data is iterated for only five times. You can increase the number of network iterations to check the effect.

– Model Summary

```
model.summary()
```

Output:

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 28, 28, 32)	832
max_pooling2d_4 (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_5 (Conv2D)	(None, 14, 14, 64)	18,496
max_pooling2d_5 (MaxPooling2D)	(None, 7, 7, 64)	0
dropout_4 (Dropout)	(None, 7, 7, 64)	0
flatten_2 (Flatten)	(None, 3136)	0
dense_8 (Dense)	(None, 128)	401,536
dropout_5 (Dropout)	(None, 128)	0
dense_9 (Dense)	(None, 10)	1,290

Total params: 1,266,464 (4.83 MB)

Trainable params: 422,154 (1.61 MB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 844,310 (3.22 MB)

– Evaluating the CNN Model

Code:

```
test_loss,test_acc=model.evaluate(x=X_test,y=y_test)
print("Test Accuracy %.2f"%test_acc)
```

Output:

```
10000/10000 [=====] - 2s 185us/sample - loss: 0.0239 - acc: 0.9921
Test Accuracy 0.99
```

The evaluation shows that the accuracy of the CNN model reaches up to 99%.

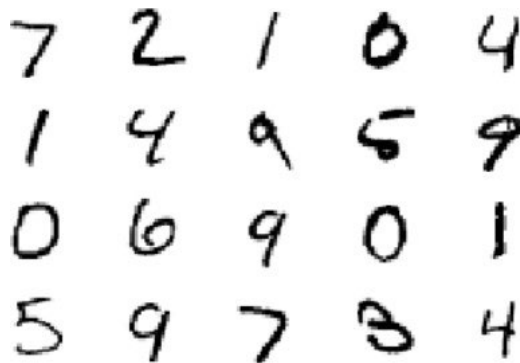
Step 4: Prediction Result Visualization

```
# Visualize the test dataset output.
import matplotlib.pyplot as plt
%matplotlib inline
def res_Visual(n):
    final_opt_a=model.predict(X_test[0:n])# Perform predictions on the test dataset by using the model.
    fig, ax = plt.subplots(nrows=int(n/5),ncols=5 )
    ax = ax.flatten()
    print('prediction results of the first {} images:'.format(n))
    for i in range(n):
        print(final_opt_a[i],end=',')
        if int((i+1)%5) ==0:
            print('\n')
        # Display images.
        img = X_test[i].reshape( (28,28))# Read each row of data in Narray format.
        plt.axis("off")
        ax[i].imshow(img, cmap='Greys', interpolation='nearest')# Visualization
        ax[i].axis("off")
    print('first {} images in the test dataset are in the following format:'.format(n))

res_Visual(20)
```

Output:

1/1 ————— 0s 46ms/step
prediction results of the first 20 images:
.....
first 20 images in the test dataset are in the following format:



7	2	1	0	4
1	4	9	5	9
0	6	9	0	1
5	9	7	3	4