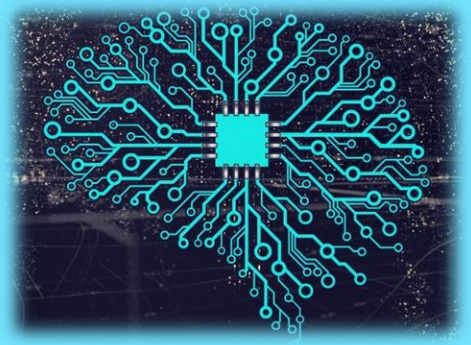
KUSE
FACULTY OF SCIENCE AND ENGINEERING
KASETSART UNIVERSITY



Chapter 3
ความรู้เบื้องต้นเกี่ยวกับคอมพิวเตอร์วิชัน
(Introduction to Computer Vision)

สอนโดย

ผู้ช่วยศาสตราจารย์ ศุษมา โชคเพิ่มพูน

การรู้จำเครื่องแต่งกาย

เราจะสามารถสร้างโมเดลให้สามารถรู้จำเสื้อผ้าและเครื่องแต่งกายได้อย่างไร

รูปตัวอย่างเสื้อผ้าและเครื่องแต่งกาย

การรู้จำเครื่องแต่งกาย

เมื่อดูภาพนี้ จะสามารถตีความได้ว่า นี่คือ เสื้อเชิ้ต รองเท้า กระเป๋า แต่ลองให้เขียนโปรแกรมตามกฎ (rule) เราต้องเขียนโปรแกรมอย่างไร?

เราอาจจะถึงทางตัน เพราะบางสิ่ง เราก็ไม่สามารถใช้ กฎ ในการอธิบายมันได้

การรู้จำเครื่องแต่งกาย

คำถาม นิสิตคิดว่าคอมพิวเตอร์วิชัน สามารถจดจำรายการเสื้อผ้า และเครื่องแต่งกายเหล่านี้ได้หรือไม่

คำตอบ

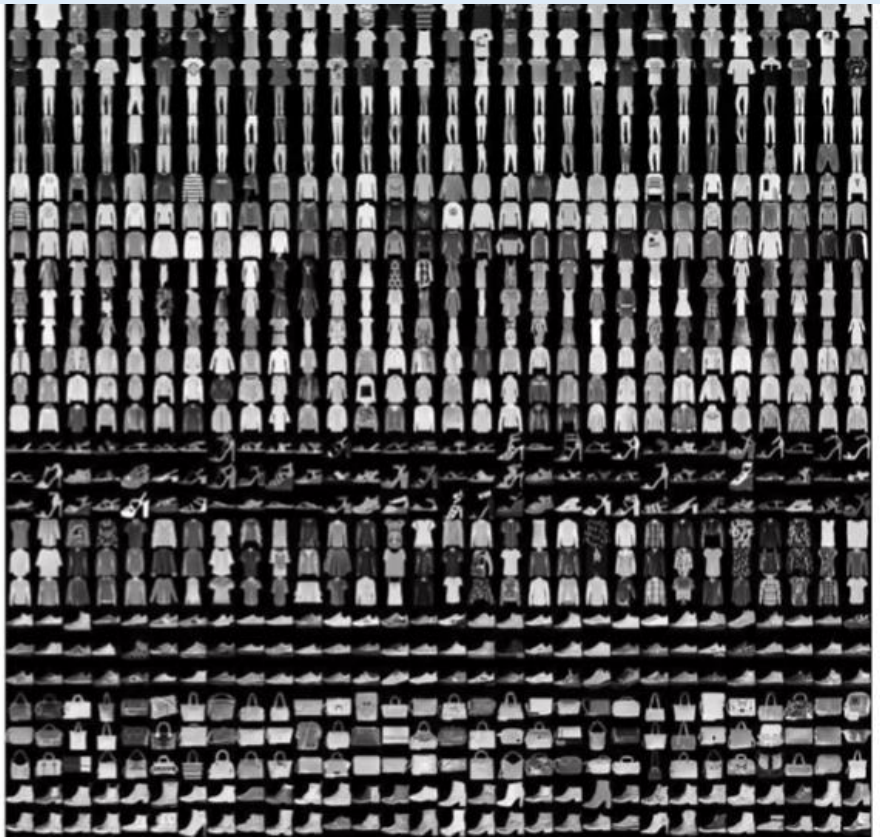
โดยการใช้ภาพ เสื้อผ้าและเครื่องแต่งกายจำนวนมาก เพื่อบอกคอมพิวเตอร์เรียนรู้ว่านั่นคือภาพอะไร จากนั้นให้คอมพิวเตอร์คิดรูปแบบที่ให้ความแตกต่าง ระหว่างรองเท้า เสื้อเชิ้ต กระเป๋าถือ เสื้อโค้ช นั่นคือสิ่งที่คุณจะได้เรียนรู้วิธีทำในส่วนนี้

ชุดข้อมูล Fashion MNIST

การฝึกสอนหรือเทรนโมเดล ด้วยข้อมูลดาต้าเซตขนาดใหญ่ ด้วยดาต้าเซตมาตารฐานที่ชื่อว่า MNIST (Modified National Institute of standard and Technology)

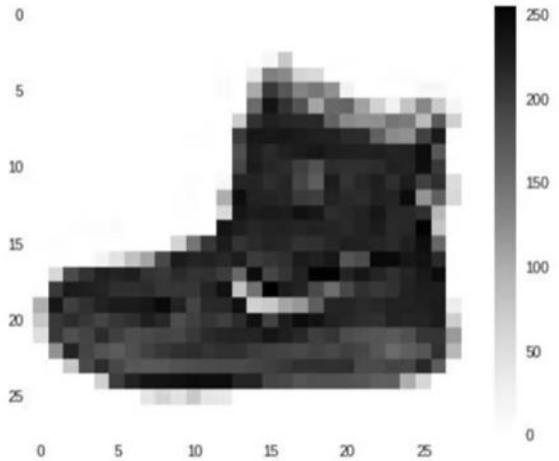
Fashion MNIST

- 70k Images
- 10 Categories
- Images are 28x28
- Can train a neural net!



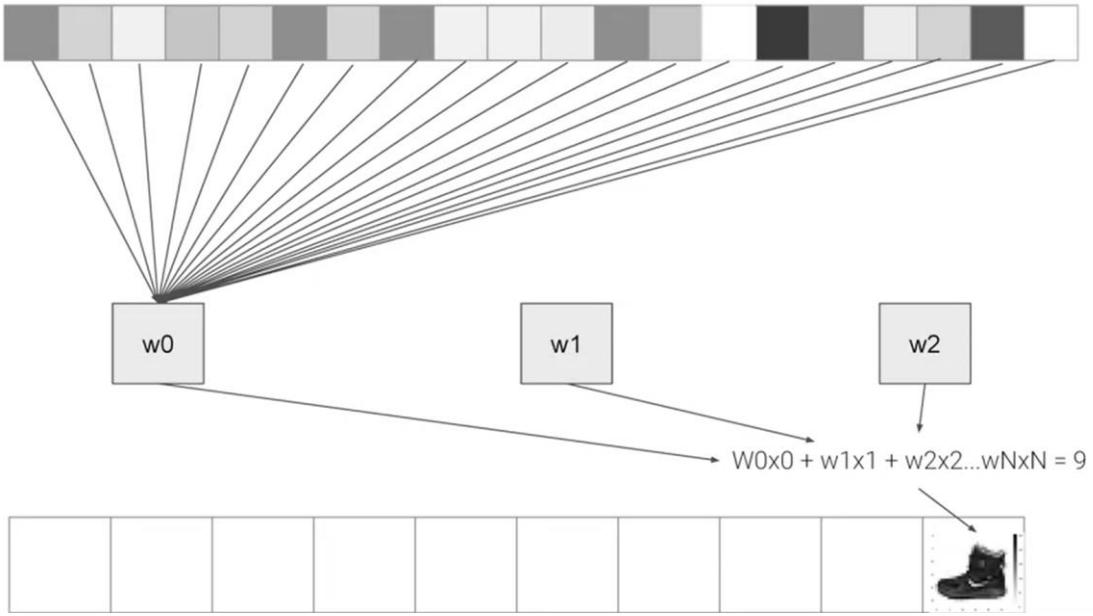
ชุดข้อมูล Fashion MNIST

ข้อมูลภาพดิจิทัล ภาพรองเท้าบูทขนาดเป็น 28x28 พิกเซลซึ่งแต่ละเม็ดพิกเซลสามารถแสดงในค่าตั้งแต่ 0 -255 โดยข้อมูลภาพจะเก็บเป็นตารางเมตริกมีขนาดเท่ากับขนาดภาพ เช่นภาพนี้โทนสีเทา ขนาด 28 x 28 พิกเซลในภาพจำเป็นต้องใช้เพียง 784 ไบต์ในการจัดเก็บทั้งภาพ



ชุดข้อมูล Fashion MNIST

ข้อมูลภาพดิจิทัล ภาพรองเท้าบูทขนาดเป็น 28x28 พิกเซล ซึ่งแต่ละเม็ดพิกเซลสามารถแสดงในค่าตั้งแต่ 0 -255 โดยข้อมูลภาพจะเก็บเป็นตารางเมตริกมีขนาดเท่ากับขนาดภาพ เช่นภาพนี้โทนสีเทา ขนาด 28 x 28 พิกเซลในภาพจำเป็นต้องใช้เพียง 784 ไบต์ในการจัดเก็บทั้งภาพ



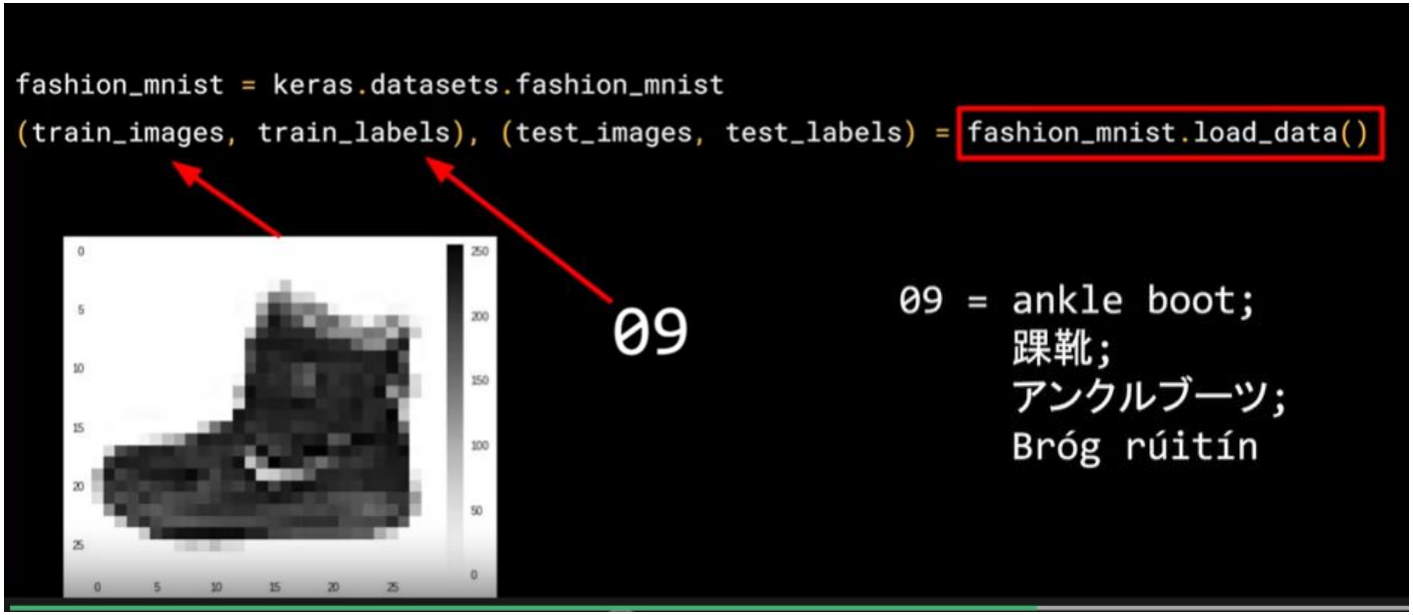
Writing code to load training data

Keras มี build-indatasets เช่น ชุดข้อมูล fashion_mnist เราสามารถใช้คำสั่ง load_data เพื่อเตรียมดาต้าเซตชุดฝึกสอน และชุดทดสอบได้ง่ายๆ ดังนี้

```
import tensorflow as tf
from tensorflow import keras



fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

การแบ่งชุดข้อมูล

ข้อมูล Fashion-MNIST ได้รับการออกแบบมาให้มีภาพสำหรับชุดข้อมูลฝึกสอน (Training) จำนวน 60,000 ภาพ และ ภาพสำหรับชุดข้อมูลทดสอบ (Testing) จำนวน 10,000 ภาพ จากภาพทั้งหมด 70,000 ภาพ



ภาพนี้แสดงเป็นเกรย์สเกลขนาด 28x28 พิกเซล และติดป้าย (label) เป็นตัวเลข การใช้ตัวเลขเพื่อหลีกเลี่ยงอคติแทนที่จะติดป้ายด้วยคำในภาษาเฉพาะ

การสร้าง Computer Vision ด้วย Neural Network

งานของเราคือการสร้างและเทรนโมเดล เพื่อให้นิวรอลเน็ตเวิร์คหาสูตรความสัมพันธ์ระหว่าง Train_images (X) และ Train_Label (Y) ปรับให้ฟิตโมเดลเพื่อหาความสัมพันธ์ที่ดีที่สุด

ส่วน Test_images /Test_Label เป็นข้อมูลที่โมเดลยังไม่เคยเห็นมาก่อน มีไว้เพื่อทดสอบประสิทธิภาพของโมเดลที่เราเทรนหรือสอนมาเพื่อให้ทราบถึงความแม่นยำของโมเดล

Coding a Computer Vision Neural Network for Classification Fashion MNIST

Import and load the Fashion MNIST data

You can access the Fashion MNIST directly from TensorFlow. Import and load the Fashion MNIST data directly from TensorFlow:

import

```
[1]  1  import tensorflow as tf
     2  print(tf.__version__)
```

```
2.7.0
```

▾ Load the Fashion MNIST dataset

แบ่งชุดข้อมูล สำหรับฝึกสอนและทดสอบ (Data Train/ Data Test)

```
[2]  1  fashion_mnist = tf.keras.datasets.fashion_mnist
```

0s

```
[3]  1  (train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

Import and load the Fashion MNIST data

The images are 28x28 NumPy arrays, with pixel values ranging from 0 to 255. The labels are an array of integers, ranging from 0 to 9. These correspond to the class of clothing the image represents:

Label	Class
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Class names in the Fashion MNIST data

Each image is mapped to a single label. Since the class names are not included with the dataset, store them here to use later when plotting the images:

```
[6]  1  class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
     2                 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

3.1 Basic classification: Classify images of clothing

Exploration data

Let's explore the format of the dataset before training the model. The following shows there are 60,000 images in the training set, with each image represented as 28 x 28 pixels

Let’s try for Exploration Test image

Train Image

```
[12]  1  train_images.shape
```

```
(60000, 28, 28)
```

Likewise, there are 60,000 labels in the training set:

```
[13]  1  len(train_labels)
```

```
60000
```

Each label is an integer between 0 and 9:

```
1  train_labels
```

```
array([9, 0, 0, ..., 3, 0, 5], dtype=uint8)
```

3.1 Basic classification: Classify images of clothing

Exploration data

Let’s try for Exploration Test image

Test image

```
[ ]  1 # Let's try for test image
```

```
(10000, 28, 28)
```

And the test set contains 10,000 images labels:

```
[ ]  1 # Let's try for test image
```

```
10000
```

```
1 # Let's try for test image
```

```
array([9, 2, 1, ..., 8, 1, 5], dtype=uint8)
```

Ex_01 What does these values look like? **

```
[7]  1  import numpy as np
     2  import matplotlib.pyplot as plt
     3  np.set_printoptions(linewidth=200)
```

```
     1  print(train_images[0]) # pixel values
```

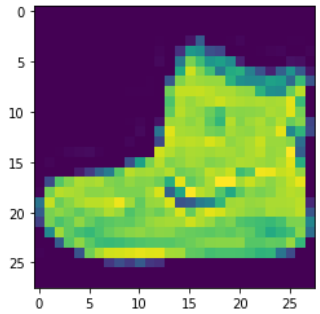
```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   1   0   0  13  73   0   0   1   4   0   0   0   0   1   1   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3   0  36 136 127  62  54   0   0   0   1   3   4   0   0   3]
 [  0   0   0   0   0   0   0   0   0   0   0   0   6   0 102 204 176 134 144 123  23   0   0   0   0  12  10   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 155 236 207 178 107 156 161 109  64  23  77 130  72  15]
 [  0   0   0   0   0   0   0   0   0   0   0   1   0  69 207 223 218 216 216 163 127 121 122 146 141  88 172  66]
 [  0   0   0   0   0   0   0   0   0   1   1   1   0 200 232 232 233 229 223 223 215 213 164 127 123 196 229   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 183 225 216 223 228 235 227 224 222 224 221 223 245 173   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 193 228 218 213 198 180 212 210 211 213 223 220 243 202   0]
 [  0   0   0   0   0   0   0   0   0   1   3   0  12 219 220 212 218 192 169 227 208 218 224 212 226 197 209  52]
 [  0   0   0   0   0   0   0   0   0   0   6   0  99 244 222 220 218 203 198 221 215 213 222 220 245 119 167  56]
 [  0   0   0   0   0   0   0   0   0   4   0   0  55 236 228 230 228 240 232 213 218 223 234 217 217 209  92   0]
 [  0   0   1   4   6   7   2   0   0   0   0   0 237 226 217 223 222 219 222 221 216 223 229 215 218 255  77   0]
 [  0   3   0   0   0   0   0   0   0  62 145 204 228 207 213 221 218 208 211 218 224 223 219 215 224 244 159   0]
 [  0   0   0   0  18  44  82 107 189 228 220 222 217 226 200 205 211 230 224 234 176 188 250 248 233 238 215   0]
 [  0  57 187 208 224 221 224 208 204 214 208 209 200 159 245 193 206 223 255 255 221 234 221 211 220 232 246   0]
 [  3 202 228 224 221 211 211 214 205 205 205 220 240  80 150 255 229 221 188 154 191 210 204 209 222 228 225   0]
 [ 98 233 198 210 222 229 229 234 249 220 194 215 217 241  65  73 106 117 168 219 221 215 217 223 223 224 229  29]
 [ 75 204 212 204 193 205 211 225 216 185 197 206 198 213 240 195 227 245 239 223 218 212 209 222 220 221 230  67]
 [ 48 203 183 194 213 197 185 190 194 192 202 214 219 221 220 236 225 216 199 206 186 181 177 172 181 205 206 115]
 [  0 122 219 193 179 171 183 196 204 210 213 207 211 210 200 196 194 191 195 191 198 192 176 156 167 177 210  92]
 [  0   0  74 189 212 191 175 172 175 181 185 188 189 188 193 198 204 209 210 210 211 188 188 194 192 216 170   0]
 [  2   0   0   0  66 200 222 237 239 242 246 243 244 221 220 193 191 179 182 182 181 176 166 168  99  58   0   0]
 [  0   0   0   0   0   0   0  40  61  44  72  41  35   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]]
```

Let's print a training image, and a training label

Ex_01 : Answer

Answer_EX_01

```
1  plt.imshow(train_images[0])  # actual graphic
2  plt.show()
```




```
[9]  1  print(train_labels[0])
```

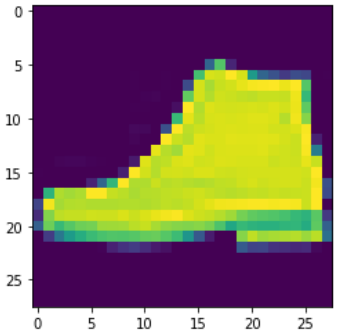
```
9
```

'Ankle boot'

EX_02 Take a look at index 42

Experiment with different indices in the array. For example, also take a look at index 42...that's a a different boot than the one at index 0

```
1  plt.imshow(train_images[42])  #42
2  plt.show()
```



Answer EX_02

Let's print a training image, and a training label

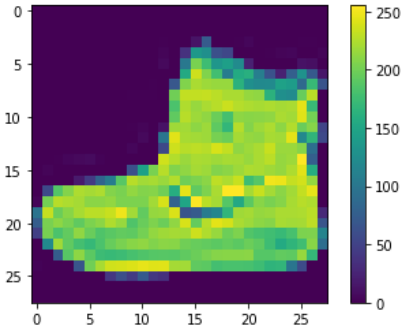
For example, index 42.

```
1  print(train_images[42]) # pixel values    # 42
2  print(train_labels[42])   # 42
3
4  plt.imshow(train_images[42])
5  plt.show()
```

Preprocess the data

The data must be preprocessed before training the network. If you inspect the first image in the training set, you will see that the pixel values fall in the range of 0 to 255:

```
1 plt.imshow(train_images[0])
2 plt.colorbar()
3 plt.show()
```



Normalization

Scale these values to a range of 0 to 1 before feeding them to the neural network model. To do so, divide the values by 255. It's important that the *training set* and the *testing set* be preprocessed in the same way:

```
[15]  1  train_images  = train_images / 255.0
      2  test_images = test_images / 255.0
```

Ex_03 Let's print a training image after normalized

```
1   print(train_images[0])
```

```
[[0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.00392157 0.         0.         0.05098039 0.28627451
  0.         0.         0.00392157 0.01568627 0.         0.         0.         0.         0.00392157 0.00392157 0.        ]
 [0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.01176471 0.         0.14117647 0.53333333 0.49803922
  0.24313725 0.21176471 0.         0.         0.         0.00392157 0.01176471 0.01568627 0.         0.         0.01176471]
 [0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.02352941 0.         0.4        0.8        0.69019608
  0.5254902  0.56470588 0.48235294 0.09019608 0.         0.         0.         0.         0.04705882 0.03921569 0.        ]
 [0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.60784314 0.9254902  0.81176471
  0.69803922 0.41960784 0.61176471 0.63137255 0.42745098 0.25098039 0.09019608 0.30196078 0.50980392 0.28235294 0.05882353]
 [0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.00392157 0.         0.27058824 0.81176471 0.8745098  0.85490196
  0.84705882 0.84705882 0.63921569 0.49803922 0.4745098  0.47843137 0.57254902 0.55294118 0.34509804 0.6745098  0.25882353]
 [0.         0.         0.         0.         0.         0.         0.         0.         0.         0.00392157 0.00392157 0.00392157 0.         0.78431373 0.90980392 0.90980392 0.91372549
  0.89803922 0.8745098  0.8745098  0.84313725 0.83529412 0.64313725 0.49803922 0.48235294 0.76862745 0.89803922 0.        ]
 [0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.71764706 0.88235294 0.84705882 0.8745098
  0.89411765 0.92156863 0.89019608 0.87843137 0.87058824 0.87843137 0.86666667 0.8745098  0.96078431 0.67843137 0.        ]
 [0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.75686275 0.89411765 0.85490196 0.83529412
  0.77647059 0.70588235 0.83137255 0.82352941 0.82745098 0.83529412 0.8745098  0.8627451  0.95294118 0.79215686 0.        ]
 [0.         0.         0.         0.         0.         0.         0.         0.         0.         0.00392157 0.01176471 0.         0.04705882 0.85882353 0.8627451  0.83137255 0.85490196
  0.75294118 0.6627451  0.89019608 0.81568627 0.85490196 0.87843137 0.83137255 0.88627451 0.77254902 0.81960784 0.20392157]
 [0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.02352941 0.         0.38823529 0.95686275 0.87058824 0.8627451  0.85490196
  0.79607843 0.77647059 0.86666667 0.84313725 0.83529412 0.87058824 0.8627451  0.96078431 0.46666667 0.65490196 0.21960784]
 [0.         0.         0.         0.         0.         0.         0.         0.         0.         0.01568627 0.         0.         0.21568627 0.9254902  0.89411765 0.90196078 0.89411765
  0.94117647 0.90980392 0.83529412 0.85490196 0.8745098  0.91764706 0.85098039 0.85098039 0.81960784 0.36078431 0.        ]
 [0.         0.         0.00392157 0.01568627 0.02352941 0.02745098 0.00784314 0.         0.         0.         0.         0.         0.92941176 0.88627451 0.85098039 0.8745098  0.87058824
  0.85882353 0.87058824 0.86666667 0.84705882 0.8745098  0.89803922 0.84313725 0.85490196 1.         0.30196078 0.        ]
 [0.         0.01176471 0.         0.         0.         0.         0.         0.         0.         0.24313725 0.56862745 0.8        0.89411765 0.81176471 0.83529412 0.86666667 0.85490196
  0.81568627 0.82745098 0.85490196 0.87843137 0.8745098  0.85882353 0.84313725 0.87843137 0.95686275 0.62352941 0.        ]
 [0.         0.         0.         0.         0.07058824 0.17254902 0.32156863 0.41960784 0.74117647 0.89411765 0.8627451  0.87058824 0.85098039 0.88627451 0.78431373 0.80392157 0.82745098
  0.90196078 0.87843137 0.91764706 0.69019608 0.7372549  0.98039216 0.97254902 0.91372549 0.93333333 0.84313725 0.        ]
 [0.         0.22352941 0.73333333 0.81568627 0.87843137 0.86666667 0.87843137 0.81568627 0.8        0.83921569 0.81568627 0.81960784 0.78431373 0.62352941 0.96078431 0.75686275 0.80784314
  0.8745098  1.         1.         0.86666667 0.91764706 0.86666667 0.82745098 0.8627451  0.90980392 0.96470588 0.        ]
 [0.01176471 0.79215686 0.89411765 0.87843137 0.86666667 0.82745098 0.82745098 0.83921569 0.80392157 0.80392157 0.80392157 0.8627451  0.94117647 0.31372549 0.58823529 1.         0.89803922
  0.86666667 0.7372549  0.60392157 0.74901961 0.82352941 0.8        0.81960784 0.87058824 0.89411765 0.88235294 0.        ]
 [0.38431373 0.91372549 0.77647059 0.82352941 0.87058824 0.89803922 0.89803922 0.91764706 0.97647059 0.8627451  0.76078431 0.84313725 0.85098039 0.94509804 0.25490196 0.28627451 0.41568627
  0.45882353 0.65882353 0.85882353 0.86666667 0.84313725 0.85098039 0.8745098  0.8745098  0.87843137 0.89803922 0.11372549]
 [0.29411765 0.8        0.83137255 0.8        0.75686275 0.80392157 0.82745098 0.88235294 0.84705882 0.7254902  0.77254902 0.80784314 0.77647059 0.83529412 0.94117647 0.76470588 0.89019608
  0.96078431 0.9372549  0.8745098  0.85490196 0.83137255 0.81960784 0.87058824 0.8627451  0.86666667 0.90196078 0.2627451 ]
 [0.18823529 0.79607843 0.71764706 0.76078431 0.83529412 0.77254902 0.7254902  0.74509804 0.76078431 0.75294118 0.79215686 0.83921569 0.85882353 0.86666667 0.8627451  0.9254902  0.88235294
  0.84705882 0.78039216 0.80784314 0.72941176 0.70980392 0.69411765 0.6745098  0.70980392 0.80392157 0.80784314 0.45098039]
 [0.         0.47843137 0.85882353 0.75686275 0.70196078 0.67058824 0.71764706 0.76862745 0.8        0.82352941 0.83529412 0.81176471 0.82745098 0.82352941 0.78431373 0.76862745 0.76078431
  0.74901961 0.76470588 0.74901961 0.77647059 0.75294118 0.69019608 0.61176471 0.65490196 0.69411765 0.82352941 0.36078431]
 [0.         0.         0.29019608 0.74117647 0.83137255 0.74901961 0.68627451 0.6745098  0.68627451 0.70980392 0.7254902  0.7372549  0.74117647 0.7372549  0.75686275 0.77647059 0.8
  0.81960784 0.82352941 0.82352941 0.82745098 0.7372549  0.7372549  0.76078431 0.75294118 0.84705882 0.66666667 0.        ]
 [0.00784314 0.         0.         0.         0.25882353 0.78431373 0.87058824 0.92941176 0.9372549  0.94901961 0.96470588 0.95294118 0.95686275 0.86666667 0.8627451  0.75686275 0.74901961
  0.70196078 0.71372549 0.71372549 0.70980392 0.69019608 0.65098039 0.65882353 0.38823529 0.22745098 0.         0.        ]
 [0.         0.         0.         0.         0.         0.         0.         0.15686275 0.23921569 0.17254902 0.28235294 0.16078431 0.1372549  0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.        ]]
```

Create Model

The first layer in this network, “tf.keras.layers.Flatten”, transforms the format of the images from a two-dimensional array (of 28 by 28 pixels) to a one-dimensional array (of 28 * 28 = 784 pixels).

After the pixels are flattened, the network consists of a sequence of two “tf.keras.layers.Dense” layers. These are densely connected, or fully connected, neural layers.

The first `Dense` layer has 128 nodes (or neurons). The second (and last) layer returns a logits array with length of 10. Each node contains a score that indicates the current image belongs to one of the 10 classes.

```
[17]  1  model = tf.keras.Sequential([
      2      tf.keras.layers.Flatten(input_shape=(28, 28)),
      3      tf.keras.layers.Dense(128, activation='relu'),
      4      tf.keras.layers.Dense(10,activation='softmax')
      5  ])
```

Compile the model

Before the model is ready for training, it needs a few more settings. These are added during the model's

- **Loss function** —This measures how accurate the model is during training. You want to minimize this function to "steer" the model in the right direction.
- **Optimizer** —This is how the model is updated based on the data it sees and its loss function.
- **Metrics** —Used to monitor the training and testing steps. The following example uses accuracy, the fraction of the images that are correctly classified.

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics =['accuracy'])
```

Train model

by calling **model.fit ** asking it to fit your training data to your training labels -- i.e. have it figure out the relationship between the training data and its actual labels, so in future if you have data that looks like the training data, then it can make a prediction for what that data would look like.

```
1  model.fit(train_images, train_labels, epochs=10)
```

Train model

by calling **model.fit ** asking it to fit your training data to your training labels -- i.e. have it figure out the relationship between the training data and its actual labels, so in future if you have data that looks like the training data, then it can make a prediction for what that data would look like.

```
1  model.fit(train_images, train_labels, epochs=10)
```

Evaluate accuracy

compare how the model performs on the test dataset:

```
1  model.evaluate(test_images, test_labels)
```

Exercise 1:

For this first exercise run the below code: It creates a set of classifications for each of the test images, and then prints the first entry in the classifications. The output, after you run it is a list of numbers. Why do you think this is, and what do those numbers represent?

```
1  classifications = model.predict(test_images)
2
3  print(classifications[0])
```

```
[1.7640241e-06 1.7171317e-08 5.5233841e-07 2.1782890e-08 9.4573309e-08 3.8364816e-03 3.8593504e-07 7.7823056e-03 1.2214412e-08 9.8837835e-01]
```

E1Q1: What does this list represent?

1. It's 10 random meaningless values
2. It's the first 10 classifications that the computer made
3. It's the probability that this item is each of the 10 classes

Answer :E1Q1

The correct answer is (3)

E1Q2: How do you know that this list tells you that the item is an ankle boot?

1. There's not enough information to answer that question
2. The 10th element on the list is the biggest, and the ankle boot is labelled 9
3. The ankle boot is label 9, and there are 0->9 elements in the list

Answer :E1Q1

The correct answer is (2)

Exercise 2:

Let's now look at the layers in your model. Experiment with different values for the dense layer with 512 neurons. What different results do you get for loss, training time etc.? Why do you think that's the case?

```
model = tf.keras.Sequential([
                    tf.keras.layers.Flatten(input_shape=(28,28)),
                    tf.keras.layers.Dense(128,activation='relu'),   #Try experimenting with this layer  #512,1024
                    tf.keras.layers.Dense(10,activation='softmax')
])
```

Let’s try !!!

E2Q1: Increase to 512 Neurons -- What's the impact?

1. Training takes longer, but is more accurate
2. Training takes longer, but no impact on accuracy
3. Training takes the same time, but is more accurate

Answer :E2Q1

The correct answer is (1)

Exercise 3:

E3Q1: What would happen if you remove the Flatten() layer. Why do you think that's the case?

```
model = tf.keras.Sequential([
                    tf.keras.layers.Flatten(input_shape=(28,28)), #Try removing this layer
                    tf.keras.layers.Dense(512,activation='relu'),
                    tf.keras.layers.Dense(10,activation='softmax')
])
```

Let’s try !!!

Answer :E3

You get an error about the shape of the data. It may seem vague right now, but it reinforces the rule of thumb that the first layer in your network should be the same shape as your data. Right now our data is 28x28 images, and 28 layers of 28 neurons would be infeasible, so it makes more sense to 'flatten' that 28,28 into a 784x1. Instead of writng all the code to handle that ourselves, we add the Flatten() layer at the begining, and when the arrays are loaded into the model later, they'll automatically be flattened for us.

Exercise 4:

Consider the final (output) layers. Why are there 10 of them? What would happen if you had a different amount than 10? For example, try training the network with 5.

```
model = tf.keras.Sequential([
                    tf.keras.layers.Flatten(input_shape=(28,28)),
                    tf.keras.layers.Dense(512,activation='relu'),
                    tf.keras.layers.Dense(10,activation='softmax')  # 5 Try experimenting with this layer
])
```

Let's try !!!

Answer :E4

You get an error as soon as it finds an unexpected value. Another rule of thumb -- the number of neurons in the last layer should match the number of classes you are classifying for. In this case it's the digits 0-9, so there are 10 of them, hence you should have 10 neurons in your final layer.

Exercise 5:

Consider the effects of additional layers in the network. What will happen if you add another layer between the one with 512 and the final layer with 10.

```
7   model = tf.keras.Sequential([
8                       tf.keras.layers.Flatten(input_shape=(28,28)),
9                        # Add a layer here,
10                      tf.keras.layers.Dense(256,activation='relu'),
11                      # Add a layer here,
12                      tf.keras.layers.Dense(10,activation='softmax')  # 5 Try experimenting with this layer
13  ])
14
```

Let’s try !!!

Answer :E5

There isn't a significant impact -- because this is relatively simple data. For far more complex data (including color images to be classified as flowers that you'll see in the next lesson), extra layers are often necessary.

Exercise 6:

E6Q1: Consider the impact of training for more or less epochs. Why do you think that would be the case?

- Try 15 epochs -- you'll probably get a model with a much better loss than the one with 5

- Try 30 epochs -- you might see the loss value stops decreasing, and sometimes increases.

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics ='accuracy')

model.fit(train_images,train_labels,epochs=10)  #Try 30 epochs
```

Let's try !!!

Answer :E6Q1

This is a side effect of something called 'overfitting' which you can learn about later and it's something you need to keep an eye out for when training neural networks. There's no point in wasting your time training if you aren't improving your loss, right!

Exercise 7:

Before you trained, you normalized the data, going from values that were 0-255 to values that were 0-1. What would be the impact of removing that? Here's the complete code to give it a try. Why do you think you get different results?

```
fashion_mnist = tf.keras.datasets.fashion_mnist
(train_images,train_labels),(test_images,test_labels) =fashion_mnist.load_data()

train_images =train_images/255.0  # Experiment with removing this line
test_images = test_images/255.0   # Experiment with removing this line

model = tf.keras.Sequential([
                    tf.keras.layers.Flatten(input_shape=(28,28)),
                    tf.keras.layers.Dense(128,activation='relu'),
                    tf.keras.layers.Dense(10,activation='softmax')
])
```

Let’s try !!!

Callback

การหยุดเทรนเมื่อถึงค่าที่ต้องการ หากต้องการฝึกฝนจนกว่าจะได้ค่าความแม่นยำที่ต้องการ เช่นต้องการฝึกฝนจนกว่าโมเดลจะมีความแม่นยำ 95% แต่ไม่รู้ว่าจะต้องฝึกกี่รอบทำอย่างไร

ตอบ วิธีที่ง่ายที่สุดคือ การใช้เทคนิคที่เรียกว่า Callback แจ้งกลับในขณะที่กำลังเทรน ทดลองดูจากโค้ดดังนี้

Exercise 8 : Callback

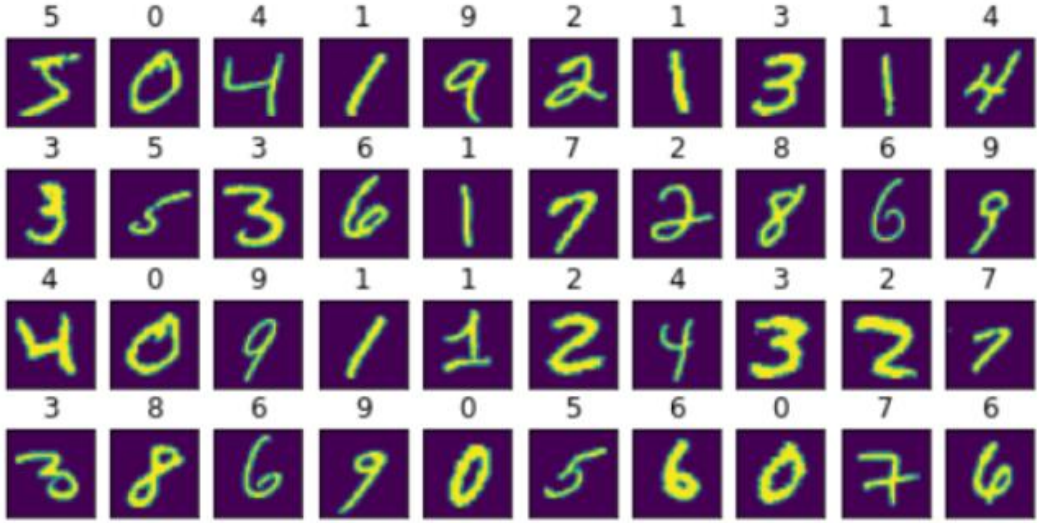
```
class myCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    if(logs.get('accuracy') >= 0.9): # Experiment with changing this value
      print("\nReached 60% accuracy so cancelling training!")
      self.model.stop_training = True

callbacks = myCallback()

fashion_mnist = tf.keras.datasets.fashion_mnist
(train_images,train_labels),(test_images,test_labels) =fashion_mnist.load_data()

train_images =train_images/255.0
test_images = test_images/255.0

model = tf.keras.Sequential([
                    tf.keras.layers.Flatten(input_shape=(28,28)),
                    tf.keras.layers.Dense(128,activation='relu'),
                    tf.keras.layers.Dense(10,activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics ='accuracy')
model.fit(train_images, train_labels, epochs=50, callbacks=[callbacks]) #50

```

Let’s try !!!

3.2 แบบทดสอบ การมองเห็นด้วยเครื่องคอมพิวเตอร์

ระบบการรู้จำลายมือ (Handwritten Digit Recognition)

ระบบรู้จำเลขลายมือเขียนซึ่งมีชื่อเรียกหลายชื่อ เช่น Handwritten Digit Recognition, Digit Recognizer, Digit Classification ฯลฯ ซึ่งโดยรวมก็คือ การแปลงภาพตัวเลขที่เขียนด้วยลายมือให้เป็นค่าตัวเลข 0-9 ที่คอมพิวเตอร์สามารถนำไปประมวลผล หรือใช้งานได้ (จาก Image> Integer)

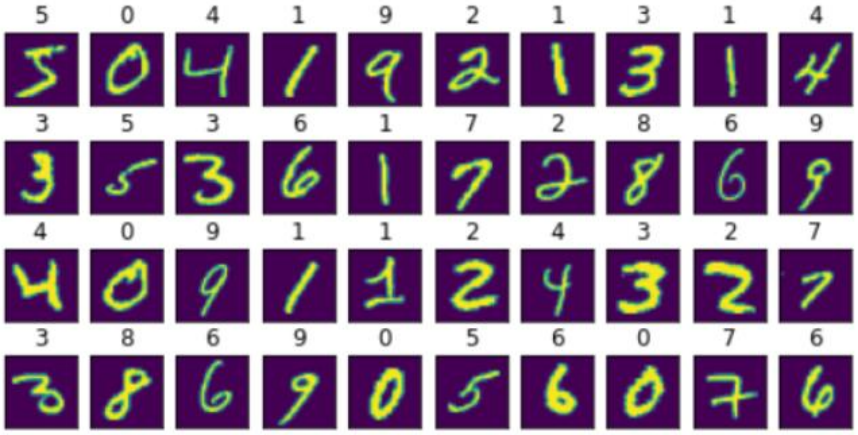


การรู้จำลายมือ (Handwritten Digit Recognition)

สามารถเอาไปประยุกต์ใช้กับงานด้านอื่นได้ เช่น ระบบตรวจจับป้ายทะเบียน ระบบตรวจจับป้ายราคา ระบบตรวจจับรหัสไปรษณีย์หน้าของจดหมายสำหรับเพื่อคัดแยกการไปรษณีย์ไปยังจังหวัดหรือเขตต่างๆ ฯลฯ

แบบทำสอบ การรู้จำลายมือ (Handwritten Digit Recognition)

หัวข้อ 3.1 เราได้เรียนรู้การทำ classificaiton โดยใช้ Fashion MNIST ซึ่งเป็นชุดข้อมูลที่มีรายการเสื้อผ้า หัวข้อ 3.2 เราจะทำการฝึกหัดกับชุดข้อมูลที่คล้ายกันอีกชุดหนึ่งที่เรียกว่า MNIST ซึ่งมีรายการเขียนด้วยลายมือตัวเลข 0 ถึง 9



แบบทำสอบ การรู้จำลายมือ (Handwritten Digit Recognition)

Write an MNIST classifier that trains to 99% accuracy or above, and does it without a fixed number of epochs -- i.e. you should stop training once you reach that level of accuracy.

แบบทดสอบ การรู้จำลายมือ (Handwritten Digit Recognition)

Some notes:

1. It should succeed in less than 10 epochs, so it is okay to change epochs= to 10, but nothing larger

2. When it reaches 99% or greater it should print out the string "Reached 99% accuracy so cancelling training!"

3. If you add any additional variables, make sure you use the same names as the ones used in the class

3.2 แบบทำสอบ Handwritten Digit Recognition

Import ♡

```
1  import tensorflow as tf
2  print(tf.__version__)
```

Load Data ♡

```
1  mnist = tf.keras.datasets.mnist
2  (train_images,train_labels),(test_images,test_labels) = mnist.load_data()
```

3.2 แบบทำสอบ Handwritten Digit Recognition

Exploration data

```
train_images.shape
```

```
train_labels.shape
```

```
test_images.shape
```

```
test_labels.shape
```

```
import numpy as np
import matplotlib.pyplot as plt

plt.imshow(train_images[0])
plt.show()
```

3.2 แบบทำสอบ Handwritten Digit Recognition

Normalization ♡

```
1  train_images = train_images/255.0
2  test_images = test_images/255.0
```

Print image after Normalization ♡

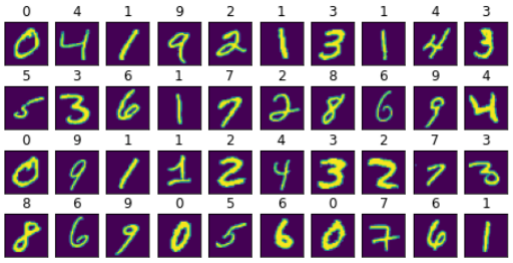
```
1  print(train_images[0])
2  plt.imshow(train_images[0])
```

3.2 แบบทำสอบ Handwritten Digit Recognition

Show feature

```
1  def visual_multi(i):
2      '''Plots 40 digits, starting with digit i'''
3      nplots = 40                                  #จำนวน Features ที่ต้องการแสดง (จำนวนกี่ภาพ)
4      fig = plt.figure(figsize=(8, 4))             #กำหนดขนาด
5      for j in range(nplots):
6          plt.subplot(4, 10, j+1)                  # 4 Row x 10 คอลัมน์
7          plt.imshow(train_images[i+j])#plt.cm.gray_r)
8          plt.title(train_labels[i+j])             #เขียนกำกับว่าเป็นเลขอะไร 0 1 2 3 .... (target/Label)
9          plt.xticks([]); plt.yticks([])
10     plt.show()
```

```
1  visual_multi(1) #แสดงข้อมูล Features ตั้งแต่ index ลำดับใดก็ให้กำหนดเลขที่ต้องการ
```



3.2 แบบทำสอบ Handwritten Digit Recognition

Create Model

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics ='accuracy')
```

Compile Model

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics ='accuracy')
```

3.2 แบบทำสอบ Handwritten Digit Recognition

Train Model ♡

```
1    model.fit(train_images,train_labels,epochs=10)
```

Test Model ♡

```
1    model.evaluate(test_images,test_labels)
```

Create Final Model

```
from sys import call_tracing
class myCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    if(logs.get('accuracy') >= 0.99): # Experiment with changing this value
      print("\nReached 60% accuracy so cancelling training!")
      self.model.stop_training = True

callbacks = myCallback()

model = tf.keras.Sequential([
                    tf.keras.layers.Flatten(input_shape=(28,28)),
                    tf.keras.layers.Dense(128,activation='relu'),
                    tf.keras.layers.Dense(10,activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics ='accuracy')
model.fit(train_images,train_labels,epochs=50,callbacks=[callbacks])

```

สรุป

บทนี้เราได้ฝึกทำโครงข่ายประสาทเทียมที่มีจำนวนมากกว่า 1 นิวรอล ได้เรียนรู้วิธีการสร้างโครงข่ายประสาทเทียมหรือนิวรอลเน็ตเวิร์ค สำหรับการมองเห็นด้วยคอมพิวเตอร์ขั้นพื้นฐาน ความสามารถของมันค่อนข้างจำกัด

เนื่องจากการเทรนด้วยข้อมูลภาพเฉดสีเทาขนาดเล็กแค่ 28 x 28 และวางเป็นระเบียบอยู่ตรงกลาง แต่ถือว่าเป็นการเริ่มต้นที่ดี สำหรับการเรียนต่อไป เราจะเรียนลึกลงไปอีกด้วยกระบวนการที่เรียกว่า Convolutions ที่คุณจะได้เรียนวิธีการสร้างโมเดลที่เข้าใจลึกลงไปในรายละเอียดของภาพ

THANK YOU

KASETSART UNIVERSITY

For Your Attention...