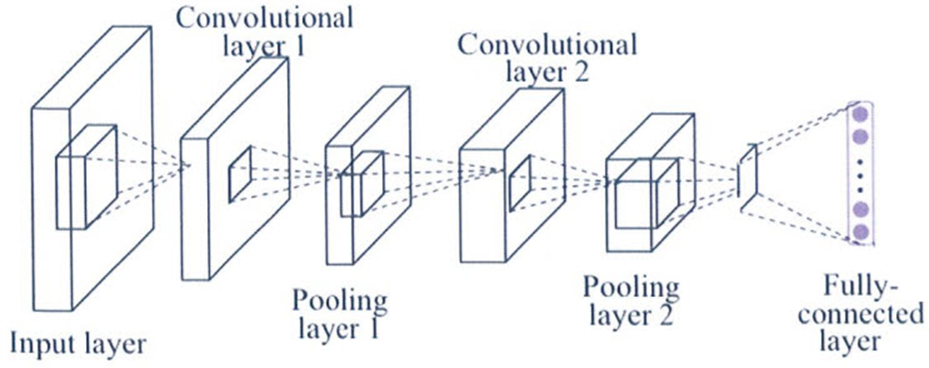
**Nhận dạng hình ảnh sử dụng máy học 2**

# 10.1. Nhận dạng số MNIST bằng CNN

Lần này, một hệ thống để phân lớp MNIST bằng các hàm cơ bản CNN của TensorFlow được xây dựng. CNN về cơ bản nhận hình ảnh và kết nối với mạng được kết nối hoàn chỉnh thông qua quá trình tích chập và lấy mẫu lặp lại. CNN được sử dụng để nhận dạng số MNIST được lập trình tương tự với cấu trúc sau:



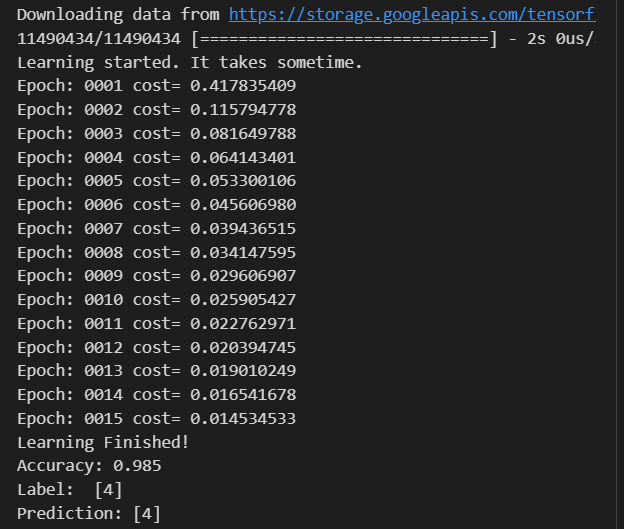
Dự đoán các số từ 0 đến 9 bằng cách kết nối kết quả của việc lặp lại lớp tích chập và lớp tổng hợp hai lần với lớp được kết nối hoàn chỉnh.

Phương pháp học và kiểm tra phía sau giống như bài học trước.

## 10.1.1. Mã nguổn chương trình đầy đủ theo phương pháp Tensorflow cũ như sau:

| import tensorflow as tf  import random  from tensorflow.keras.datasets import mnist  tf.compat.v1.disable\_eager\_execution()  tf.compat.v1.set\_random\_seed(777) # reproducibility  # Load MNIST dataset  (x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()  # Flatten the images and normalize pixel values to the range [0, 1]  x\_train = x\_train.reshape([-1, 784]) / 255.0  x\_test = x\_test.reshape([-1, 784]) / 255.0  # Convert labels to one-hot encoding  y\_train = tf.keras.utils.to\_categorical(y\_train, 10)  y\_test = tf.keras.utils.to\_categorical(y\_test, 10)  # hyper parameters  learning\_rate = 0.001  training\_epochs = 15  batch\_size = 100  # input place holders  X = tf.compat.v1.placeholder(tf.float32, [None, 784])  X\_img = tf.reshape(X, [-1, 28, 28, 1]) # img 28x28x1 (black/white)  Y = tf.compat.v1.placeholder(tf.float32, [None, 10])  # -------------- Cấu trúc tầng mạng --------------  # Lớp Convolutional Đầu Tiên  W1 = tf.Variable(tf.random.normal([3, 3, 1, 32], stddev=0.01))  L1 = tf.nn.conv2d(X\_img, W1, strides=[1, 1, 1, 1], padding='SAME')  L1 = tf.nn.relu(L1)  L1 = tf.nn.max\_pool(L1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')  # Lớp Convolutional Thứ Hai  W2 = tf.Variable(tf.random.normal([3, 3, 32, 64], stddev=0.01))  L2 = tf.nn.conv2d(L1, W2, strides=[1, 1, 1, 1], padding='SAME')  L2 = tf.nn.relu(L2)  L2 = tf.nn.max\_pool(L2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')  L2\_flat = tf.reshape(L2, [-1, 7 \* 7 \* 64])  # Lớp Fully Connected (1)  W3 = tf.Variable(tf.random.normal([7 \* 7 \* 64, 10], stddev=0.01))  b = tf.Variable(tf.random.normal([10]))  logits = tf.matmul(L2\_flat, W3) + b  cost = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(logits=logits, labels=Y))  optimizer = tf.compat.v1.train.AdamOptimizer(learning\_rate=learning\_rate).minimize(cost)  # initialize  sess = tf.compat.v1.Session()  sess.run(tf.compat.v1.global\_variables\_initializer())  # train my model  print("Learning started. It takes sometime.")  for epoch in range(training\_epochs):  avg\_cost = 0  total\_batch = int(x\_train.shape[0] / batch\_size)  for i in range(total\_batch):  batch\_xs, batch\_ys = x\_train[i \* batch\_size:(i + 1) \* batch\_size], y\_train[i \* batch\_size:(i + 1) \* batch\_size]  feed\_dict = {X: batch\_xs, Y: batch\_ys}  c, \_ = sess.run([cost, optimizer], feed\_dict=feed\_dict)  avg\_cost += c / total\_batch  print('Epoch:', '%04d' % (epoch + 1), 'cost=', '{:.9f}'.format(avg\_cost))  print("Learning Finished!")  # Test model and check accuracy  correct\_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(Y, 1))  accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))  print('Accuracy:', sess.run(accuracy, feed\_dict={X: x\_test, Y: y\_test}))  # Get one and predict  r = random.randint(0, x\_test.shape[0] - 1)  print("Label: ", sess.run(tf.argmax(y\_test[r:r + 1], 1)))  print("Prediction:", sess.run(tf.argmax(logits, 1), feed\_dict={X: x\_test[r:r + 1]})) |
| --- |

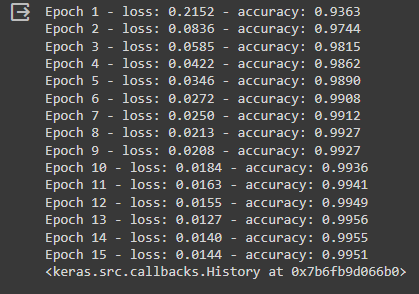
Kết quả nhận được:

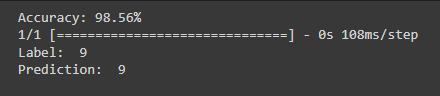


## 10.1.2. Mã nguồn chương trình đầy đủ theo phương pháp API Keras như sau:

| # Importing the required Keras modules containing model and layers  import tensorflow as tf  from tensorflow.keras import layers, models  import random  (x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.mnist.load\_data()  # Reshaping the array to 4-dims so that it can work with the Keras API  x\_train = x\_train.reshape(x\_train.shape[0], 28, 28, 1)  x\_test = x\_test.reshape(x\_test.shape[0], 28, 28, 1)  input\_shape = (28, 28, 1)  # Making sure that the values are float so that we can get decimal points after division  x\_train = x\_train.astype('float32')  x\_test = x\_test.astype('float32')  # Normalizing the RGB codes by dividing it to the max RGB value.  x\_train /= 255  x\_test /= 255  print('x\_train shape:', x\_train.shape)  print('Number of images in x\_train', x\_train.shape[0])  print('Number of images in x\_test', x\_test.shape[0])  # Create a sequential model  model = models.Sequential()  # -------------- Cấu trúc tầng mạng --------------  # Lớp Convolutional Đầu Tiên  model.add(layers.Conv2D(28, kernel\_size=(3, 3), input\_shape=input\_shape))  model.add(layers.MaxPooling2D(pool\_size=(2, 2)))  # Lớp Fully Connected (1)  model.add(layers.Flatten())  model.add(layers.Dense(128, activation=tf.nn.relu))  model.add(layers.Dropout(0.2))  # Lớp Fully Connected (2)  model.add(layers.Dense(10, activation=tf.nn.softmax))  model.compile(optimizer='adam',  loss='sparse\_categorical\_crossentropy',  metrics=['accuracy'])  # Custom callback to format output  class CustomCallback(tf.keras.callbacks.Callback):  def on\_epoch\_begin(self, epoch, logs=None):  print(f'Epoch {epoch + 1}', end=' ')  def on\_epoch\_end(self, epoch, logs=None):  print(f'- loss: {logs["loss"]:.4f} - accuracy: {logs["accuracy"]:.4f}')  # Create an instance of the custom callback  custom\_callback = CustomCallback()  # Fit the model with the custom callback  model.fit(x=x\_train, y=y\_train, epochs=15, callbacks=[custom\_callback], verbose=0)  # serialize model to JSON  model\_json = model.to\_json()  with open("model.json", "w") as json\_file:  json\_file.write(model\_json)  # serialize weights to HDF5  model.save\_weights("model.h5")  print("Saved model to disk")  # Tải mô hình đã lưu  json\_file = open('model.json', 'r')  loaded\_model\_json = json\_file.read()  json\_file.close()  loaded\_model = tf.keras.models.model\_from\_json(loaded\_model\_json)  # Tải trọng số vào mô hình đã tải  loaded\_model.load\_weights("model.h5")  # Biên soạn lại mô hình đã tải  loaded\_model.compile(optimizer='adam',  loss='sparse\_categorical\_crossentropy',  metrics=['accuracy'])  # Chuẩn bị dữ liệu kiểm thử  (x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.mnist.load\_data()  x\_test = x\_test.reshape(x\_test.shape[0], 28, 28, 1)  x\_test = x\_test.astype('float32') / 255  # Đánh giá mô hình trên dữ liệu kiểm thử  accuracy = loaded\_model.evaluate(x\_test, y\_test, verbose=0)[1]  print('Accuracy: {:.2%}'.format(accuracy))  # Lấy một ví dụ ngẫu nhiên và in ra nhãn thực tế và nhãn dự đoán  random\_index = random.randint(0, len(x\_test) - 1)  label = y\_test[random\_index]  prediction = loaded\_model.predict(x\_test[random\_index].reshape(1, 28, 28, 1))  predicted\_label = tf.argmax(prediction, 1).numpy()[0]  print("Label: ", label)  print("Prediction: ", predicted\_label) |
| --- |

Nếu chạy chương trình này, sẽ thấy kết quả sau:





Có thể mất một chút thời gian tùy thuộc vào hiệu suất của PC cho đến khi kết quả được đưa ra và độ chính xác được cải thiện khoảng 99%. Nó khác biệt một chút so với kết quả trước đó, nhưng khi tiến gần hơn 100%, thật khó để cải thiện độ chính xác, đó là một bước tiến đáng kể.

# 10.2. So sánh CNN theo cách Tensorflow cũ và API Keras

## 10.2.1. So sánh tổng quan

Đoạn mã đầu tiên và đoạn mã thứ hai đều triển khai mô hình **Convolutional Neural Network** (CNN) để phân loại chữ số viết tay từ tập dữ liệu MNIST, nhưng chúng sử dụng các thư viện khác nhau và có một số khác biệt. Dưới đây là so sánh giữa chúng:

|  | | **Tensorflow gốc** | **API Keras** |
| --- | --- | --- | --- |
| **Giống** | **Tải dữ liệu** | Sử dụng tập dữ liệu MNIST để phân loại chữ số viết tay  Chúng tiền xử lý dữ liệu bằng cách chuẩn hóa giá trị pixel về khoảng [0, 1]. | |
| **Kiến trúc mô hình** | CNN và bao gồm các lớp convolutional đi sau là các lớp max-pooling và các lớp fully connected | |
| **Huấn luyện** | Sử dụng trình tối ưu hóa Adam và mất mát cross-entropy (hoặc sparse categorical cross-entropy trong mã Keras) | |
| **Lưu mô hình** | Lưu mô hình đã được huấn luyện vào đĩa | |
| **Khác** | **Thư viện** | Tensorflow trực tiếp | API Keras (API cấp cao được tích hợp vào tensorflow) |
| **Cấu trúc mã** | Định nghĩa biến, phép toán và phiên chi tiết hơn.  Sử dụng các cấu trúc Tensorflow cấp thấp hơn để định nghĩa và chạy mô hình | Sử dụng API Sequential của Keras, giảm bớt chi tiết và tạo ra mã nguồn ngắn gọn và dễ đọc hơn. |
| **Batch Normalization** | Không bao gồm chuẩn hóa theo batch | Không bao gồm trực tiếp nhưng có thể dễ dàng thêm vào bằng cách sử dụng lớp **BatchNormalization** |
| **Dropout** | Bao gồm dropout trong lớp fully connected | Bao gồm dropout sau lớp dense đầu tiên |
| **Hàm kích hoạt** | Sử dụng hàm kích hoạt ReLU cho các lớp ẩn | Sử dụng ReLU cho các lớp dense và softax cho lớp đầu ra |
| **Lưu mô hình** | Lưu bằng 2 tệp riêng việt: “model.json” cho kiến trúc và “model.h5” cho trọng số | Kết hợp kiến trúc và trọng số cho 1 tệp duy nhất “model.h5” |

⇒ Tóm lại, cả hai đoạn mã đều đạt được mục tiêu phân loại chữ số MNIST bằng CNN, nhưng đoạn mã thứ hai sử dụng Keras cung cấp biểu diễn của mô hình ngắn gọn và cao cấp hơn, làm cho mã nguồn dễ hiểu và dễ duy trì hơn.

## 10.2.2. So sánh Cấu trúc tầng mạng

Tính đến cấu trúc mạng, đây là những khác biệt chính giữa hai mô hình:

| **Đặc điểm** | **Mô hình TensorFlow** | **Mô hình Keras** |
| --- | --- | --- |
| Lớp Convolutional thứ nhất | Kernel size(3, 3), 32 filters, ReLU, Max-pooling(2, 2) | Kernel size (3, 3), 28 filters, ReLU, Max-pooling(2, 2) |
| Lớp Convolution thứ hai | Kernel size (3, 3) 64 filters, ReLU, Max-pooling (2, 2) | - |
| Lớp Fully Connected 1 | 3136 neurons (7 \* 7 \* 64) | 128 neurons, ReLU |
| Lớp Dropout 1 | - | Dropout (tỷ lệ 0.2) |
| Lớp Fully Connected 2 | 10 neurons, không có hàm kích hoạt | 10 neurons, Softmax |

**Chú ý:**

· *Cả hai mô hình sử dụng hàm kích hoạt ReLU cho lớp convolutional và fully connected (trừ lớp output trong mô hình Keras).*

· *Mô hình TensorFlow sử dụng hai lớp convolutional đầu tiên trước fully connected, trong khi mô hình Keras chỉ sử dụng một lớp convolutional và có một lớp dropout để ngăn chặn overfitting*

· *Lớp dropout trong mô hình Keras được thêm vào để giảm overfitting.*

· *Mô hình Keras sử dụng hàm kích hoạt Softmax cho lớp output để tính xác suất của từng lớp.*