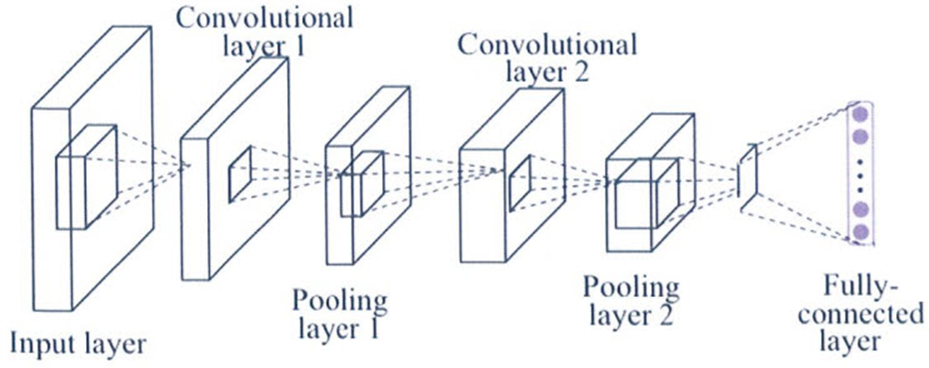
# **Nhận dạng hình ảnh sử dụng máy học 2**

## **10.2 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:



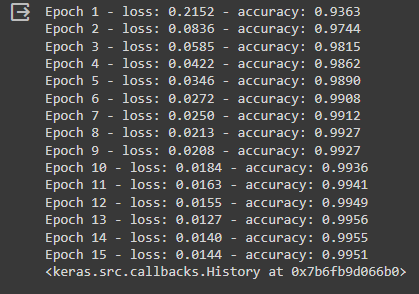
Dự đoán các số từ 0 đến 9 bằng cách kết nối kết quả của việc lặp lại lớp tích chập và lớp tổng hợp hai lần với lớp được kết nối hoàn chỉnh.

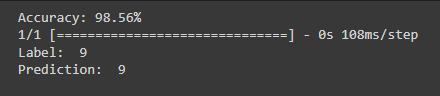
Phương pháp học và kiểm tra phía sau giống như bài học trước.

Mã nguồn chương trình đầy đủ 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.