**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**

**BỘ MÔN**: KHOA HỌC MÁY TÍNH

**AI SYSTEM**

**HANDWRITTING RECOGNITION USING TENSORFLOW**

**Giáo viên hướng dẫn:**

Nguyễn Ngọc Thảo

**Thông tin sinh viên:**

**Tên:** Bùi Trọng Xuyến

**Mssv:** 1612835

**Email:** btxuyenhcmus@gmail.com

**Tên:** Lương Công Văn

**Mssv:** 1612804

**Email:** congvan2498@gmail.com

**Tên:** Phạm Hoàng Tín

**Mssv:** 1612696

**Email:** Phamhoangtin1998@gmail.com

**tp. HCM, 26/04/2019**

1. **GIỜI THIỆU**

* Nội dung: thiết lập và huấn luyện một mô hình nhận dạng ký tự viết tay (handwritting recognition).
* Hệ thống là một mạng kết hợp neural network (cNN) dưới sự hỗ trợ của thư viện tensorflow. Nhập dạng được thành 47 kiểu ký tự bao gồm bảng chữ cái viết thường, bảng chữ cái viết hoa, và 10 số tự nhiên.
* MNIST special database 19:
* Chứa toàn bộ tài liệu đào tạo của NIST cho tài liệu in dấu tay và nhận dạng ký tự. Nó xuất bản với mẫu dấu tay từ 3600 nhà văn, 810,000 hình ảnh ký tự được phân lập từ biểu mẫu của họ.
* Version của bộ dữ liệu trong hệ thống lần này được lấy từ nguồn sau:

[*https://s3.amazonaws.com/nist-srd/SD19/by\_merge.zip*](https://s3.amazonaws.com/nist-srd/SD19/by_merge.zip)

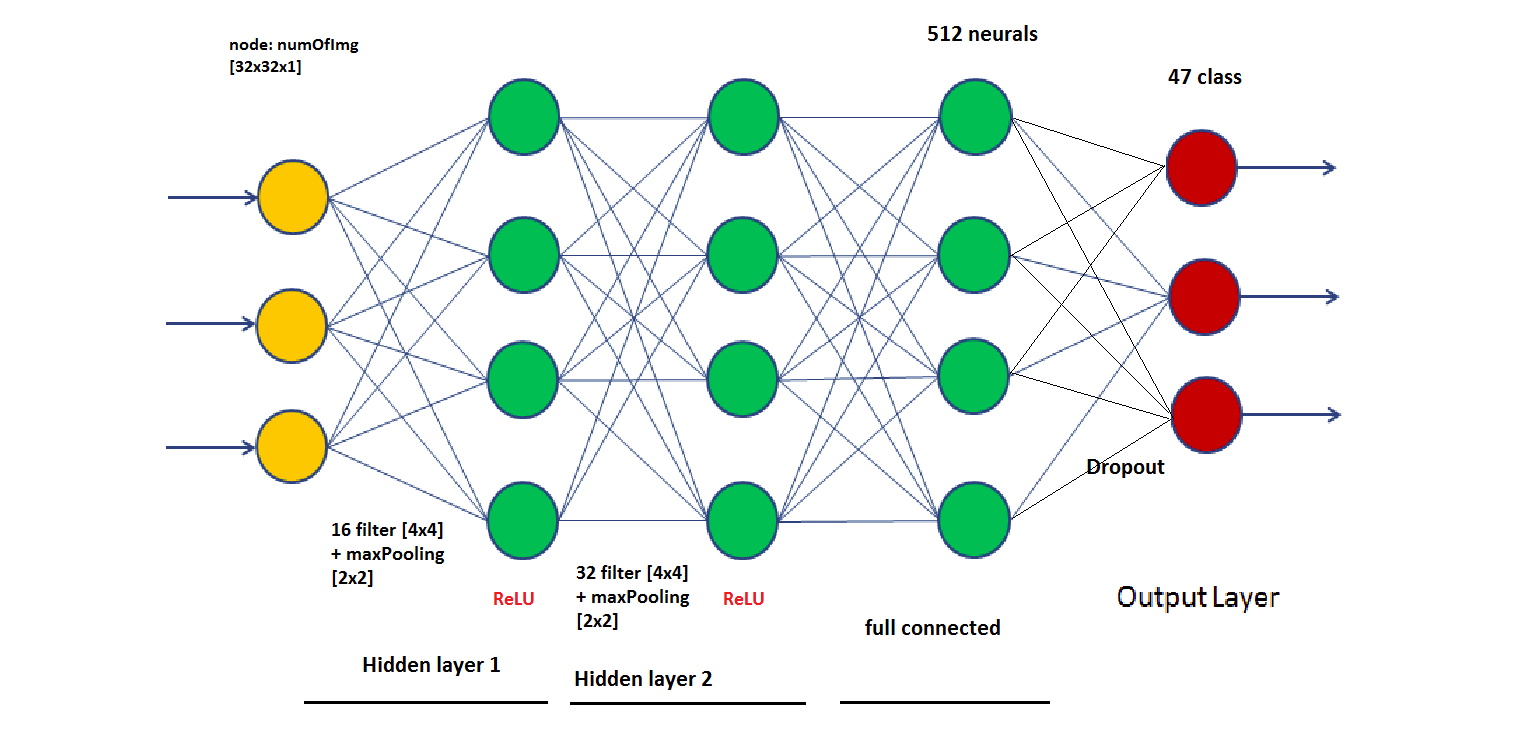
“Dung lượng 530.013 KB”

* Convolutional neural network (CNN):
* Giống với mạng neural thông thường, chúng được tạo từ các neural(*Node*) có trọng lượng (*w*). Mỗi neural sẽ nhận được một giá trị đầu vào và thực hiện “function activation”. Toàn bộ mạng theo một sơ đồ duy nhất, từ các pixel thô ban đầu sẽ qua các “hidden layer” với các trọng số khác nhau, số lượng “hidden layer” càng nhiều hay độ phức tạp của mạng càng nhiều thì độ chính xác càng cao, sau đó sẽ được phân lớp và cho ra output đúng.
* Mạng neural nhận được một đầu vào và biến đổi nó thông qua một loạt các “hidden layer”. Một “hidden layer” được tạo ra từ một tập hợp các neural, trong đó một neural được kết nối hoàn toàn (hoặc một số) neural ở lớp trước, mỗi neural trong một lớp hoạt động độc lập và không chia sẻ tài nguyên cho bất kì neural nào trong lớp đó.



***Hình 1.*** *Neural nerwork*

* Sơ đồ khối hệ thống xây dựng lần này



***Hình 2:*** *sơ đồ khối hệ thống*

1. **PREPROCESSING THE DATA**

* Tải xuống cơ sỡ dữ liệu “by\_merge.zip” sau đó đặt chúng vào cùng một thư mục (data) và đổi tên tất cả theo một định dạng “class\_[class]\_index\_[index].png” (class\_47 \_index\_001.png).
* Chạy tập lệnh trong file “rename\_images.py”.

*copyfile(os.path.join(subdir, file),*

*os.path.join(output\_dir, "class\_{}\_index\_{}.png".format(class\_index, index)))*

* Sử dụng “opencv” để chuyển đổi những hình ảnh đã theo cấu trúc quy định này thành 2 mảng. Một mảng chứa các hình ảnh, được định kích cỡ lại thành 32x32 và thang độ xám với dạng (num\_images, 32, 32, 1). Mảng khác chứa label của các hình ảnh đó có dạng (num\_labels, 47).
* Chạy tập lệnh “data\_handler.py”.
* Trong tập lệnh này chúng ta cần sử dụng tới thư viện “opencv-python” để thao tác với hình ảnh.

*C:\Users\Administrator>pip install opencv-python*

Hoặc

*C:\Users\Administrator>pip install opencv-python==”version”*

Sẽ có 2 trường hợp xảy ra là máy bạn đã tồn tại opencv hoặc là nó sẽ tải các gói site-packages hỗ trợ opencv về và bạn có thể sử dụng nó ngay với câu lệnh

*>>import cv2 as cv*

* Tạo mảng thứ nhất

*def* *get\_2d\_array(im\_path):*

*"""Return image scaled with factor 0.5 as numpy array."""*

*im\_color = cv2.imread(im\_path) # Read the image as a numpy array.*

*im\_color = im\_color[32:64 + 32, 32:64 + 32]*

*# Shape = (64,64,3) (x\_pixels,y\_pixels, color\_channels)*

*im\_color = cv2.resize(im\_color, (32,32)) # Rescale the image*

*# Shape = (32,32,3) (x\_pixels\*scale, y\_pixels\*scale, color\_channels)*

*im = np.zeros(shape=(32,32,1)) # Create an empty array with the final shape (32,32,1)*

*for i, x in enumerate(im\_color): # Fill the array*

*for n, y in enumerate(x): # Note: We cannot use cv2.cvtColor(im\_color, cv2.COLOR\_BGR2GRAY), because*

*im[i][n][0] = (y[0] + y[1] + y[2]) / 3 # that will return an array with the shape (32,32), but we need*

*return im*

* Tạo mảng thứ hai

*def get\_label(name):*

*"""Returns label number for file name*

*>>> get\_label("class\_10\_Index\_3454.jpeg")*

*10*

*"""*

*return int(name.split("\_")[1])*

* Để thực thi được 2 hàm này thì chúng ta phải cung cấp đối số cho 2 hàm này chính là đường dẫn thư mục ảnh mà chúng ta đã đối tên lại và cho chung vào một thư mục như đã yêu cầu ở trên. Và cách đọc là kỹ thuật của ngôn ngữ python.
* Như vậy là ta có thể kết thúc bước tiền xử lý dữ liệu và 2 mảng đã tạo đó sẽ trực tiếp tham gia vào CNN.

Tổng quan lại như sau

* Tải database về và giải nén.
* Chạy tệp “rename\_images.py” để tổng hợp tất cả các hình ảnh về một thư mục với cấu trúc tên quy định.
* Chạy tệp “data\_handler.py” để tạo 2 mảng tham gia vào phần sau.

1. **TRAINING DATA AND TEST**

* Xác định lại sơ đồ mô hình của chúng ta:

1. Convolution layer: filter\_size 4x4, filters: 16
2. Pooling layer 2x2
3. Convolution layer: filter\_size 4x4, filters: 32
4. Pooling layer 2x2
5. Densely connected layer: neurons: 512
6. Dropout
7. Output layer: neurons: 47

* Các kỹ thuật bên trong được thể hiện trong tập lệnh “training\_32x32.py”.
* Import

*import numpy as np*

*import tensorflow as tf*

*import math*

*import sys*

*import time*

*import datetime*

*import os*

Chúng ta cần “numpy” để làm việc với dữ liệu mảng, “tensorflow” sẽ là thành phần chính vì chúng ta đang sử dụng nó để tạo ra AI, “math” để tính toán các công thức gọn gàng, “sys” để đọc các đối số-sử dụng nó để chỉ định số lượng chu kỳ đào tạo thông qua dòng lệnh, “datetime” được sử dụng để ghi nhật kí.

* Install tensorflow and run tensorflow

*> conda create –n tensorflow python=”version”*

*> activate tensorflow*

*> pip install –ignore-installed –upgrade tensorflow*

Kiểm thử:

*> python*

*>> import tensorflow as tf*

*>> tf.\_\_version\_\_*

* Tạo một lớp helper để làm việc với tập train và test:

*class NISTHelper():*

*def \_\_init\_\_(self, train\_img, train\_label, test\_img, test\_label):*

*self.i = 0*

*self.test\_i = 0*

*self.training\_images = train\_img*

*self.training\_labels = train\_label*

*self.test\_images = test\_img*

*self.test\_labels = test\_label*

i và test\_i dùng để theo dõi chỉ số hiện tại trong mảng train và mảng test.

training\_images và test\_images là mảng với cấu trúc [?, 32, 32, 1], ? là số lượng hình ảnh, 32, 32 tương ứng với độ phân giải của hình ảnh và 1 tương đương với số kênh màu thì ở đây được xử lý ở mức xám nên số kênh là 1.

trainning\_labels và test\_labels là mảng với cấu trúc [?, 47], ngày từ đầu chúng ta đã cho 47 nhãn dán khác nhau.

*def next\_batch(self, batch\_size):*

*x = self.training\_images[self.i:self.i + batch\_size]*

*y = self.training\_labels[self.i:self.i + batch\_size]*

*self.i = (self.i + batch\_size) % len(self.training\_images)*

*return x, y*

hàm này trả về hình ảnh batch\_size tiếp theo.

* Triển khai từng lớp trong mô hình
* Các hàm trong quá trình huấn luyện

*def unison\_shuffled\_copies(a, b):*

*assert len(a) == len(b), "Độ dài 2 mảng không bằng nhau!"*

*p = np.random.permutation(len(a))*

*return a[p], b[p]*

Cứ mỗi lần lặp lại epochs ta thực hiện việc xáo trộn lại tập train.

*def init\_weights(shape):*

*init\_random\_dist = tf.truncated\_normal(shape, stddev=0.1)*

*return tf.Variable(init\_random\_dist)*

Khởi tạo w với kích thước shape theo phân phối chuẩn

*def init\_bias(shape):*

*init\_bias\_vals = tf.constant(0.1, shape=shape)*

*return tf.Variable(init\_bias\_vals)*

Khởi tạp bias với ky vọng là 0.1.

*def conv2d(x, W):*

*return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')*

Tích chập cho x với trọng số W có độ dịch chuyển stride = 1, có thêm padding.

*def max\_pool\_2by2(x):*

*return tf.nn.max\_pool(x, ksize=[1, 2, 2, 1],*

*strides=[1, 2, 2, 1], padding='SAME')*

Lấy max trong vùng cửa sổ ksize với độ dịch chuyển strides

*def convolutional\_layer(input\_x, shape, name="unspecified"):*

*with tf.name\_scope(name):*

*with tf.name\_scope("weights"):*

*W = init\_weights(shape)*

*variable\_summaries(W)*

*with tf.name\_scope("biases"):*

*b = init\_bias([shape[3]])*

*with tf.name\_scope("Wx\_plus\_b"):*

*preactive = conv2d(input\_x, W) + b*

*tf.summary.histogram("pre\_activations", preactive)*

*#Kích hoạt hàm activation function, làm ReLU*

*activations = tf.nn.relu(preactive, name="activation")*

*tf.summary.histogram("activations", activations)*

*#Trả về kết quả sau khi hidden layer*

*return activations*

Sau khi khởi tạo W và bias, ta thực hiện phép phi tuyến y = input\_x\*W + bias. Sau đó kích hoạt hàm activation function là ReLU và trả ra kết quả

* Trình tự huấn luyện tham số

Đọc dữ liệu đã qua bước tiền xử lý, 2 tệp “nist\_images\_32x32.npy” và “nist\_labels\_32x32.npy”.

Sau đó xáo trộn dữ liệu lên

*images = np.load("nist\_images\_32x32.npy")*

*labels = np.load("nist\_labels\_32x32.npy")*

*images, labels = unison\_shuffled\_copies(images, labels)*

Dũng 70% dữ liệu trong tập MNIST để train và 30% còn lại dùng làm tập test kiểm tra độ chính xác của mô hình.

*split = math.ceil(len(images) \* 0.7)*

*train\_imgs = images[:split]*

*train\_labels = labels[:split]*

*# Tập test*

*test\_imgs = images[split:]*

*test\_labels = labels[split:]*

*log("Performed train-test split")*

Đặt 2 biến x (input) và y\_true (labels) vào trình giữ chổ

x = tf.placeholder(tf.float32, shape=[None, 32, 32, 1], name="x") *# Input, shape = ?x32x32x1*

y\_true = tf.placeholder(tf.float32, shape=[None, 47], name="y\_true") *# Labels*

Qua hidden layer 1

*convo\_1 = convolutional\_layer(x, shape=[4, 4, 1, 32], name="Convolutional\_1")*

*convo\_1\_pooling = max\_pool\_2by2(convo\_1) # shape=?x16x16x32*

Qua hidden layer 2

*convo\_2 = convolutional\_layer(convo\_1\_pooling, shape=[4, 4, 32, 64], name="Convolutional\_2")*

*convo\_2\_pooling = max\_pool\_2by2(convo\_2) # shape=?x8x8x64*

*convo\_2\_flat = tf.reshape(convo\_2\_pooling, [-1, 8\*8\*64])*

Full connected

*full\_layer\_one = normal\_full\_layer(convo\_2\_flat, 1024, tf.nn.relu, name="Normal\_Layer\_1")*

Dropout ra kết quả

*with tf.name\_scope("dropout"):*

*hold\_prob = tf.placeholder(tf.float32)*

*tf.summary.scalar("dropout\_keep\_probability", hold\_prob)*

*full\_one\_dropout = tf.nn.dropout(full\_layer\_one, keep\_prob=hold\_prob)*

*y\_pred = normal\_full\_layer(full\_one\_dropout, 47, act=tf.identity,*

*name="Output\_Layer") # Layer with 47 neurons for one-hot encoding*

*with tf.name\_scope("cross\_entropy"):*

*with tf.name\_scope("total"):*

*# Calculate cross-entropy*

*cross\_entropy = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits\_v2(labels=y\_true, logits=y\_pred))*

*tf.summary.scalar("cross\_entropy", cross\_entropy)*

*with tf.name\_scope("train"):*

*train = tf.train.AdamOptimizer(learning\_rate=0.001).minimize(cross\_entropy)*

*with tf.name\_scope("accuracy"):*

*with tf.name\_scope("correct\_predictions"):*

*correct\_predictions = tf.equal(tf.argmax(y\_pred, 1), tf.argmax(y\_true, 1)) # use argmax to get the index*

*# of the highest value in the prediction array and compare that with the true array to generate and array*

*# of the form [True,False,True]*

*with tf.name\_scope("accuracy"):*

*accuracy = tf.reduce\_mean(tf.cast(correct\_predictions, tf.float32)) # Calculate percentage of correct*

* Thực nghiệm
* Mô hình sử dụng tỷ lệ học tập là 0.002, nhưng vì thời gian và phần cứng không cho phép nên nhóm đã cho chạy trên 2000 epochs và cho ra tỷ lệ đúng là 0.78, theo nghiêm cứu thì sau 10000 epochs thì giá trị đúng không còn chênh lệch nhiều.
* Cứ 200 epochs, thử nghiệm với kích thước 200 để đánh giá độ chính xác

*if (i % 200) == 0: # Every 200 epochs evaluate with test set*

*matches = tf.equal(tf.argmax(y\_pred, 1), tf.argmax(y\_true, 1))*

*acc = tf.reduce\_mean(tf.cast(matches, tf.float32))*

*batch = nist.test\_batch(200)*

*# Hold probability is 1 to get the best results*

*summary, accuracy = sess.run([merged, acc], feed\_dict={x: batch[0], y\_true: batch[1], hold\_prob: 1})*

*test\_writer.add\_summary(summary, i) # Save the results of test batch*

*saver.save(sess, "log/model.ckpt", i) # Save model*

*if i > 0:*

*log("Step: " + str(i) + "; Accuracy: " + str(accuracy) + "; Time (200 Steps): " + str((time.time() - epoch\_start)))*

*else:*

*log("Step: " + str(i) + "; Accuracy: " + str(accuracy) + ";")*

*epoch\_start = time.time()*

* Quá trình tôi đã làm

Step 200; Accuracy: 0.12; time: 518

Step 400; Accuracy: 0.36; time: 463

Step 600; Accuracy: 0.49; time: 570

Step 800; Accuracy: 0.605; time: 570

Step 1000; Accuracy: 0.675; time: 578

Step 1200; Accuracy: 0.695; time: 477

Step 1400; Accuracy: 0.71; time: 519

Step 1600; Accuracy: 0.735; time: 558

Step 1800; Accuracy: 0.78; time: 557

* Sau khi chạy tập lệnh huấn luyện trong “file trainning\_32x32.py” chúng ta sẽ có được models lưu trong thư mục “\models” bao gồm 4 file
* 32x32\_2conv\_32\_64\_1norm\_1024.ckpt.data-00000-of-00001
* 32x32\_2conv\_32\_64\_1norm\_1024.ckpt.index
* 32x32\_2conv\_32\_64\_1norm\_1024.ckpt.meta
* checkpoint

1. **PREDICTION**

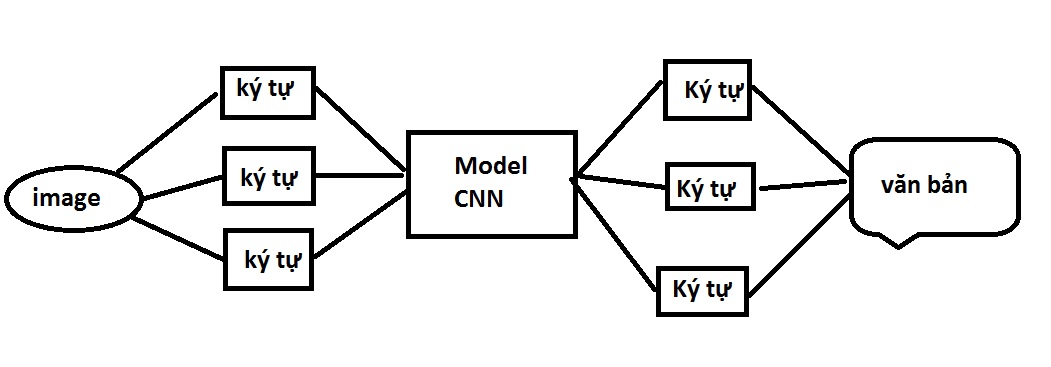
* Xây dựng lại mô hình giống như trong training\_32x32 với các hàm hỗ trợ
* Init\_weight
* Init\_bias
* Conv2d
* Max\_pool\_2by2
* Convolution\_layer
* Normal\_full\_layer
* Sự khác biệt nằm ở hàm predict(single\_image):
* Dùng opencv để đưa ảnh về mảng numpy
* Load checkpoint đã được tạo ra trước đó

checkpoint = "models/32x32\_2conv\_32\_64\_1norm\_1024.ckpt"

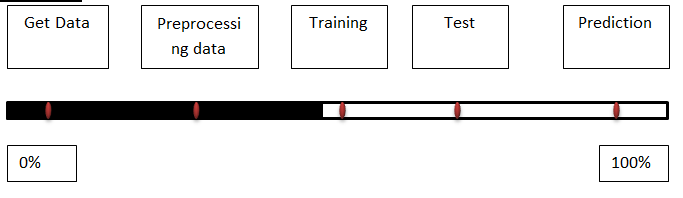
* #DICT

1 class gồm 47 phần từ là 10 chữ số và bảng chữ cái viết thường, viết hoa.

* Chạy tương đương như train, sau đó ta được giá trị preditions, tra trong class ký tự tương ứng và return
* Hiện tại mới truyền vào từng hình ký tự và xuất ra kết quả tương ứng, hướng phát triển là đưa hình văn bản và detect được cả văn bản



1. **PROCESSING**



* Bước lấy dữ liệu và tiền xử lý đã hoàn thành
* Ở bước này khó khăn chính là cài python3 trên môi trường window
* Cài opencv cho python trên môi trường window. Thường xảy ra lỗi không tương thích version opencv và không tương thích bàn 32bit hay 64bit.
* Bước training mới bước đầu giống với lý thuyết đã học, còn lại phụ thuộc vào thư viện và tài liệu trên internet.
* Gặp khó khăn với bức ảnh có kích thước lớn, chẳng hạn như văn bản dài,...

1. **REFERENCE**

[1]. [*https://www.nist.gov/srd/nist-special-database-19*](https://www.nist.gov/srd/nist-special-database-19)

[2]. [*http://cs231n.github.io/convolutional-networks/*](http://cs231n.github.io/convolutional-networks/)

[3].[*https://github.com/frereit/TensorflowHandwritingRecognition*](https://github.com/frereit/TensorflowHandwritingRecognition)