ДОДАТОК А

Графічний матеріал атестаційної роботи

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

ФАКУЛЬТЕТ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА УПРАВЛІННЯ
КАФЕДРА КІТС

Нейромережева обробка та розпізнавання об’єктів в умовах кольорової близькості

Магістрант гр. КІТм-19-1 Гнібеда А.О.
Науковий керівник проф. Руденко О. Г.

Харків 2020

Мета роботи

Метою атестаційної роботи є розробка нейромережі для обробки та розпізнавання об’єктів в умовах кольорової близькості.

Для досягнення даної мети необхідно вирішити наступні задачі:

- проаналізувати та розглянути сучасні роботи та публікації для визначення актуальності роботи;
- визначити сфери використання, переваги, нелодіки та перспективи розвитку нейронних мереж;
- розглянути сучасні методи розробки нейронних мереж;
- Створення програми для навчання моделі.

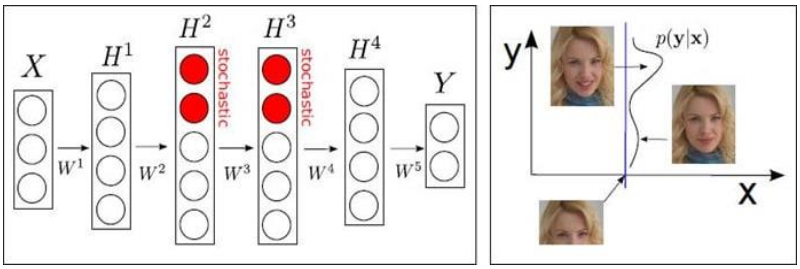
2

Тристороння категоризація

1. Глибокі мережі для неконтрольованого або генеративного навчання;
2. Глибокі мережі для контрольованого навчання;
3. Гібридні глибинні мережі

3

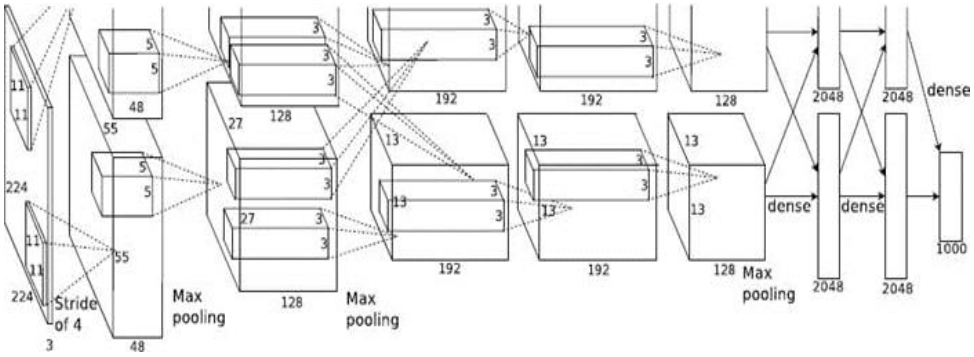
Безконтрольне або генеративне навчання особливостей



Типова архітектура стохастичної нейронної мережі з чотирма прихованими шарами

4

Навчання та класифікація ознак під наглядом



Архітектура системи глибокого CNN, яка перемогла у 2012 році у конкурсі ImageNet

5

Методи обробки зображення та їх фільтрація

1. Порогова обробка
2. Лінійні фільтри
3. Нелінійні фільтри
4. Математична морфологія
5. Метод класифікації зображення: персептрон

6

Порогова обробка

Порогова обробка – найпростіший метод сегментації чорно-білих зображень. Замінює кожен піксель зображення на білий (255), якщо значення кольору більше, ніж певний поріг. Якщо значення пікселя менше порога – колір пікселя замінюється на чорний (0). Метод також називають бінаризація.

Приклад бінаризації методом Оцу

7

Лінійні фільтри

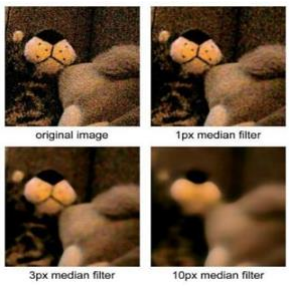
Згладжуючі фільтри роблять зображення нечіткими або розмитими. Зокрема для згладжування контурів фігур використовуються фільтр, так звана згортка. Його суть полягає в тому, що кожен піксель зображення замінюється на деяке середнє значення оточуючих його пікселів. . Кожен фільтр має своє ядро – матрицю коефіцієнтів, на яку множаться сусідні пікселі цільового зображення. Наприклад - розмиття по Гаусу. Його ядро для пікселя (m, n) розмірності (u, v) радіуса r обчислюється за формулою:

$$y(m,n) = \frac{1}{2\pi r^2} \sum_{u,v} e^{\frac{-(u^2+v^2)}{2r^2}} x(m+u, n+v) \, .$$

8

Нелінійні фільтри

Медіанна фільтрація – метод нелінійної обробки сигналів, запропонована Тьюкі. Особливо ефективний для фільтрації білого шуму. Одновимірний медіанний фільтр являє собою вікно, яке охоплює непарне число елементів, зображення. Центральний елемент замінюється медіаною всіх елементів зображення у вікні.

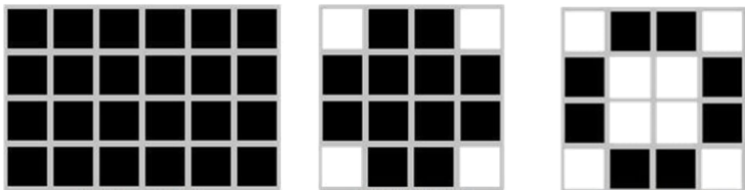


Приклад використання медіанного фільтра до зашумлення зображення з трьома різними значеннями радіусу вікна фільтрації

9

Математична морфологія

Теорія і техніка морфологічного аналізу і обробки зображень заснована на теорії множин. Розглянемо бінарну морфологію: зображення представляється у вигляді прямокутних бінарних матриць, де одиниця означає білий колір, а нуль – чорний. Для кожної морфологічної операції, так само, як і для фільтрів, необхідно ядро, яке в даному випадку називається структурним елементом .



Основні структурні елементи математичної морфології

10

Метод класифікації зображення: персептрон

Для вирішення завдання класифікації отриманих відфільтрованих зображень дуже ефективний метод класифікації, заснований на персептроні. Персептрон, в свою чергу, заснований на штучних нейронах – модель клітини мозку.

Штучний нейрон – зважений суматор, єдиний вихід якого визначається через його входи і матрицю ваг за формулою:

$$y = f(u), u = \sum_{i=1}^{n} x_i w_i + x_0 w_0 A \circ B = (A \ominus B) \oplus B$$

Навчання персептрона – обчислення таких коефіцієнтів для всіх нейронів крім вихідних, при яких значення нейронної мережі буде близькою до бажаного.

11

Технології

Для фільтрації зображень з принципу карток Ішіхара, була використана бібліотека OpenCV, а саме функції, які реалізують бінаризацію, фільтрацію і методи математичної морфології. На першому етапі проводилася бінаризація зображень.

OpenCV – бібліотека алгоритмів комп'ютерного зору з відкритим вихідним кодом. До першої версії розроблялася в Центрі розробки програмного забезпечення Intel. OpenCV написана на мові високого рівня (C/C ++, Python) і містить алгоритми для інтерпретації зображень, калібрування камери за зразком, усунення оптичних спотворень, визначення подібності, аналіз переміщення об'єкта, визначення форми об'єкту та стеження за об'єктом, 3D-реконструкції, сегментації об'єкта , розпізнавання жестів і т.д.

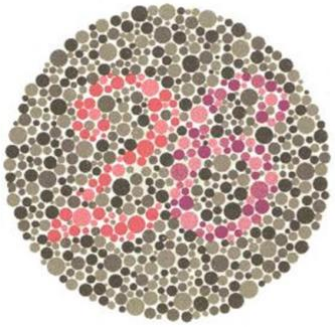
Для реалізації персептрона використовувалося готове рішення – Фреймворк keras.

Keras – це бібліотека з відкритим вихідним кодом, що дозволяє легко створювати нейронні мережі. Бібліотека сумісна з TensorFlow, Theano і іншими бібліотеками машинного навчання.

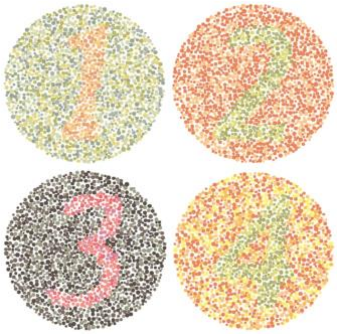
В якості алгоритму навчання використовується RMSProp –вдосконалений метод градієнтного спуску, що включає в себе метод бегущего середнього і адаптивність

12

Результати



Оригінальна картка тесту Ішіхара



Приклади згенерованих карток

13

Висновок

У роботі були розглянуті методи представлення, розпізнавання і обробки зображень, розглянуті операції математичної морфології, реалізовано додаток вирішання тесту Ішіхара. Тестування було проведено на згенерованому зашумленому наборі карт Ішіхара. Етап вибору колірних каналів, відділення цифр від фону і відновлення їх форми пройшло задовільно. Вся вибірка була розділена на тестову і навчальну в співвідношенні 600/800. В результаті навчання персептрона на навчальній вибірці, точність розпізнавання на тестових даних досягла 99%.

14

ДОДАТОК Б

Програмна реалізація

ttf2png.py:

```
import sys
import os
import subprocess
import random
from fontTools.ttLib import TTFont

RESULT_PATH = "fonts"
FONT_COUNT = 30
BAD_FONTS =

(B'cst ', b'rsfs', b'lklug ', b'esint', b'ani ', b'cmex', b'msam ',
b'EagleLake')

def gen_pics (TTF_PATH, FONT_SIZE = "500"):
'' '
Generate png picture of digits with fonts from TTF_PATH.
'' '
TTF_NAME = os.path.splitext (os.path.basename (TTF_PATH)) [0]

for i in range (10):
name = str (i)

output_png = os.path.join (RESULT_PATH, name + "_" + TTF_NAME + ".png")

subprocess.call ([ "convert-im6.q16", "-font", TTF_PATH, "-pointsize",
FONT_SIZE, "-background", "#FFFFFF", "label:" + str (i), output_png])

def main ():
# Read all fonts

process = subprocess.Popen ([ 'convert', '-list', 'font'], stdout =
subprocess.PIPE)
out = process.communicate () [0]
# Choose good fonts
out = [x.decode ( "utf-8") for x in out.split () \
if x.find (b'ttf ')! = -1 and all (map (lambda type:
x.find (type) == -1, BAD_FONTS))]
# Choise random fonts

ttfs = random.sample (out, FONT_COUNT)
print ( "Generate:")
for x in ttfs:
print (x)
'' '

ttfs = [ "/home/dupeljan/.local/share/fonts/GamjaFlower-Regular.ttf", \

"/Usr/share/fonts/truetype/tlwg/Kinnari.ttf", \ "/ usr / share / fonts /
truetype / tlwg / Laksaman-

Italic.ttf ", \
"/home/dupeljan/.local/share/fonts/Lora-
Bold.ttf ", \
```

```
"/Usr/share/fonts/truetype/malayalam/Uroob.ttf"]
'' '
if not os.path.isdir (RESULT_PATH):
os.mkdir (RESULT_PATH)
for ttf in ttfs:

gen_pics (ttf)
print ( "Generate digits successfuly")


if __name__ == '__main__':
main ()


norm.py:


import cv2

import numpy as np

import os


DIR_NAME = "fonts"

BORDER = 1.5

def gen_norm (name):

'' '

Search contour, cut out the shape

and insert into the center of the new picture.


Border is the ratio of the length of square edge

of obtained image to the larger side of the cut-out digit.

'' '


img = cv2.imread (os.path.join (DIR_NAME, name + '. png'))

hsv = cv2.cvtColor (img, cv2.COLOR_BGR2HSV)

thresh = cv2.inRange (hsv, 0, 255,0)

contours, hierarchy = cv2.findContours (thresh, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE) [1:]


try:

x0 = (min (contours [0] [i] [0] [0] for i in range (contours [0] .shape
[0])), min (contours [0] [i] [0] [1] for i in range (contours [0] .shape [0])))

x1 = (max (contours [0] [i] [0] [0] for i in range (contours [0] .shape
[0])), max (contours [0] [i] [0] [1] for i in range (contours [0] .shape [0])))

except IndexError:
```

```
        print ( "Error to norm", name)

        return

        weight = height = int (max (x1 [0] - x0 [0], x1 [1] - x0 [1]) * BORDER)

        new_img = np.zeros ((weight, height, 3), np.uint8)

        cv2.rectangle (new_img, (0,0), (weight, height), (255,255,255), - 1)

        dx = int (weight / 2 - (x1 [1] - x0 [1]) / 2)

        dy = int (height / 2 - (x1 [0] - x0 [0]) / 2)

        new_img [dx: x1 [1] -x0 [1] + dx, dy: x1 [0] - x0 [0] + dy] = img [x0
[1]: x1 [1], x0 [0]: x1 [ 0]]



        cv2.imwrite (os.path.join (DIR_NAME, name + '. png'), new_img)

        def main ():

        files = os.listdir (DIR_NAME)

        for filename in files:

        name, ext = os.path.splitext (filename)

        if ext == ".png":

        gen_norm (name)

        print ( "Normalize all pic in current dir")

        if __name__ == '__main__':

        main ()

        ishihara.py

        import math

        import random

        import os

        import sys

        from PIL import Image, ImageDraw

        SOURCE_PATCH = "fonts"

        RESULT_PATH = "test5"
```

```
GEN_COUNT = 1

try:
from scipy.spatial import cKDTree as KDTree
import numpy as np
IMPORTED_SCIPY = True
except ImportError:
IMPORTED_SCIPY = False

BACKGROUND = (255, 255, 255)
TOTAL_CIRCLES = 1500

color = lambda c: ((c >> 16) & 255, (c >> 8) & 255, c & 255)
if sys.argv [1]:
COLOR_THEAM = int (sys.argv [1])% 5 else:
COLOR_THEAM = 1

TYPE = COLOR_THEAM if COLOR_THEAM <= 3 else 3

type 1
if COLOR_THEAM == 1:
COLORS_ON = [
color (0xF9BB82), color (0xEBA170), color (0xFCCD84)
]
COLORS_OFF = [
color (0x9CA594), color (0xACB4A5), color (0xBBB964), color (0xD7DAAA),
color (0xE5D57D), color (0xD1D6AF)
]

# Type 2
elif COLOR_THEAM == 2:
COLORS_ON = [
color (0xb6b87c), color (0xe3da73), color (0xb0ab60)
]
COLORS_OFF = [
color (0xef845a), color (0xffc68c), color (0xef845a),
```

```
      ]


      # Type 3

      elif COLOR_THEAM == 3:

      COLORS_ON = [

      color (0xf79087), color (0xf26969), color (0xd8859d), \ color (0xf79087)

      ]
      COLORS_OFF = [

      color (0x5a4e46), color (0x7b6b63), color (0x9c9c84),

      ]


      # Type 3 too

      elif COLOR_THEAM == 4:

      COLORS_ON = [

      color (0xb6b87c), color (0xe3da73), color (0xb0ab60)

      ]

      COLORS_OFF = [

      color (0xef845a), color (0xffc68c), color (0xef845a), \ color (0xfff36b),
color (0xffbd52)

      ]



      def    generate_circle    (image_width,    image_height,    min_diameter,
max_diameter):

      radius = random.triangular (min_diameter, max_diameter,

      max_diameter * 0.8 + min_diameter *
      0.2) / 2


      angle = random.uniform (0, math.pi * 2)

      distance_from_center = random.uniform (0, image_width * 0.48 - radius)

      x = image_width * 0.5 + math.cos (angle) * distance_from_center y =
image_height * 0.5 + math.sin (angle) * distance_from_center

      return x, y, radius



      def overlaps_motive (image, par):

      (X, y, r) = par
```

```
      points_x = [x, x, x, xr, x + r, xr * 0.93, xr * 0.93, x + r * 0.93, x +
r * 0.93]

      points_y = [y, yr, y + r, y, y, y + r * 0.93, yr * 0.93, y + r * 0.93, yr
* 0.93]


      for xy in zip (points_x, points_y):

      try:

      if image.getpixel (xy) [: 3]! = BACKGROUND:

      return True

      except IndexError:

      print ( "Exept gen")

      return False

      except TypeError:

      print ( "second exept gen")

      return False


      return False



      def circle_intersection (par1, par2):

      (X1, y1, r1) = par1

      (X2, y2, r2) = par2

      return (x2 - x1) ** 2 + (y2 - y1) ** 2 <(r2 + r1) ** 2



      def circle_draw (draw_image, image, par):

      (X, y, r) = par

      fill_colors = COLORS_ON if overlaps_motive (image, (x, y, r)) else
COLORS_OFF

      fill_color = random.choice (fill_colors)


      draw_image.ellipse ((x - r, y - r, x + r, y + r),

      fill = fill_color,

      outline = fill_color)
```

```
def gen_test (name, gen_n = 0):
'' '
Generate Ishihara test card
from file name.png to name + gen_n + .png
'' '
image = Image.open (os.path.join (SOURCE_PATCH, name + ".png"))
image2 = Image.new ( 'RGB', image.size, BACKGROUND)
draw_image = ImageDraw.Draw (image2)

width, height = image.size

min_diameter = (width + height) / 200
max_diameter = (width + height) / 75

circle = generate_circle (width, height, min_diameter, max_diameter)
circles = [circle]

circle_draw (draw_image, image, circle)

try:
for i in range (TOTAL_CIRCLES):
tries = 0
if IMPORTED_SCIPY:
kdtree = KDTree ([(x, y) for (x, y, _) in circles])
while True:
circle = generate_circle (width, height, min_diameter, max_diameter)
elements, indexes = kdtree.query ([(circle [0],
circle [1])], k = 12)
for element, index in zip (elements [0],
indexes [0]):
if not np.isinf (element) and
circle_intersection (circle, circles [index]):
break
else:
break
```

```
tries + = 1
else:
while any (circle_intersection (circle, circle2) for circle2 in circles):
tries + = 1
circle = generate_circle (width, height, min_diameter, max_diameter)

#print ( '{} / {} {}'. format (i, TOTAL_CIRCLES, tries))

circles.append (circle)
circle_draw (draw_image, image, circle)
except (KeyboardInterrupt, SystemExit):
pass

name = name + "theme_" + str (COLOR_THEAM) + "type_" + str (TYPE) + ".png"
image2.save (os.path.join (RESULT_PATH, name), "PNG")
print ( "Generate" + name + "succesfully")

def main ():
files = os.listdir (SOURCE_PATCH)
if not os.path.isdir (RESULT_PATH):
os.mkdir (RESULT_PATH)
count = 0
for filename in files:
name, ext = os.path.splitext (filename)
if ext == ".png":
for i in range (GEN_COUNT):
gen_test (name, i)
count + = 1
print ( "not more than" + str (len (files) - count) + "files left")
print ( "Generate test success")

if __name__ == '__main__':
main ()
```

noise.py:

```
import cv2

import numpy as np

from random import randint

import os


SOURCE_PATCH = "test5"

RESULT_PATH = "noise test5"


def add_noise (name, k = 10):

'' '

Add noise in each pixel of name.png.

256 / k - ratio of dispersion.

'' '

range_ = int (512 / k)

img = cv2.imread (os.path.join (SOURCE_PATCH,   name + ".png"))

for i in range (len (img)):

for j, x in enumerate (img [i]):

rand = [int (randint (0, range_) - range_ / 2) for i in
range (3)]

for k in range (3):

sum_ = x [k] + rand [k]

if sum_ <= 0:

rand [k] = 0

elif 0 <sum_ <255:

rand [k] = sum_

else:

rand [k] = 255

img [i] [j] =     rand

cv2.imwrite (os.path.join (RESULT_PATH, name + ". png"), img)

print (name + ". png" + "generated")


def main ():

if not os.path.isdir (RESULT_PATH):

os.mkdir (RESULT_PATH)
```

```
files = os.listdir (SOURCE_PATCH)

count = 0

for filename in files:

name, ext = os.path.splitext (filename)

if ext == ".png":

add_noise (name)

count + = 1

print ( "not more than" + str (len (files) - count) + "files

left ")

print ( "Generate test success")


if __name__ == '__main__':

main ()

filter.py:

import cv2
import numpy as np

import os

SOURCE_PATCH = "noise test5"
RESULT_PATH = "filtred_noise 5"

def filter (name):
 '' '

Select number from name.png test card.
Create b / w picture of it.
'' '
rgb = cv2.imread (os.path.join (SOURCE_PATCH, name + '. png'))
lab = cv2.cvtColor (rgb, cv2.COLOR_BGR2Lab)

average_a = np.array (list (x [1] for i in range (len (lab)) for x in lab
[i])) .mean ()
average_b = 0

a = cv2.split (lab) [1]

if average_a <130: # MANY GREEN OR BLUE, IN A CONTRAST rgb = cv2.threshold
(a, 132,255, cv2.THRESH_BINARY) [1] state = 1

else:

average_b = np.array (list (x [2] for i in range (len (lab)) for x in lab
[i])) .mean ()

if average_b <135: # LITTLE RED => RED CHANNEL DARK, IN A CONTRAST

state = 2
rgb =
```

```
      cv2.threshold (a, 0,255, cv2.THRESH_BINARY + cv2.THRESH_OTSU) [1] # else:

      state = 3 # БАГАТО ЧЕРВОНОГО => MANY RED => RED CHANNEL LIGHT, IN RED
CONTRAST
      rgb =

      cv2.threshold (cv2.split (rgb) [2], 227,255, cv2.THRESH_BINARY) [1]
      rgb = cv2.bitwise_not (rgb)

      kernel_opening = np.ones ((2,2), np.uint8)
      kernel_closing = np.ones ((12, 12), np.uint8)

      opening = cv2.morphologyEx (rgb, cv2.MORPH_OPEN, kernel_opening) closing
= cv2.morphologyEx (opening, cv2.MORPH_CLOSE,
      kernel_closing)

      dst = cv2.medianBlur (closing, 5)

      cv2.imwrite (os.path.join (RESULT_PATH, name + ". png"), dst)
      print (name + ". png filtered")


      def main ():
      if not os.path.isdir (RESULT_PATH):
      os.mkdir (RESULT_PATH)
      files = os.listdir (SOURCE_PATCH)

      for filename in files:
      name, ext = os.path.splitext (filename)
      if ext == ".png":
      filter (name)

      if __name__ == '__main__':
      main ()
      network.py
      from keras.utils import to_categorical

      from keras import models

      from keras import layers
      import numpy as np
      import os
      import cv2

      TEST_PATH = "filtred_noise 5"
      DATASET_PATH = TEST_PATH + "_ dataset" + ".npz"
      TRAIN_COUNT = 800

      def create_dataset ():
      '' '
      Generate BORDER in file DATASET_PATH.npz element "dataset"
      '' '

      images = list ()
      labels = list ()
      files = os.listdir (TEST_PATH)
      for filename in files:
      name, ext = os.path.splitext (filename)
      if ext == ".png":
      img = cv2.imread (os.path.join (TEST_PATH, name + '. png'))
      img = np.compress ([True], cv2.resize (img, (28,
      28)). Reshape (784,3), axis = 1)
      images.append (img)
      labels.append (int (name [0]))
```

```
      np.savez (DATASET_PATH, dataset = np.array ([images, labels]))

      def get_dataset (path = DATASET_PATH):
      '' '
      Generate BORDER from file DATASET_PATH.npz
      '' '
      data = np.load (path) [ 'dataset']
      return (np.stack (data [0], axis = 0), np.stack (data [1], axis = 0))


      def create_network ():
      images, labels = get_dataset ()

      train_images, test_images = images [: TRAIN_COUNT], images [TRAIN_COUNT:]

      train_labels, test_labels = labels [: TRAIN_COUNT], labels [TRAIN_COUNT:]

      network = models.Sequential ()

      network.add (layers.Dense (512, activation = 'relu', input_shape = (28 *
28,)))
      network.add (layers.Dense (10, activation = 'softmax'))

      network.compile      (optimizer      =      'rmsprop',      loss      =
'categorical_crossentropy', metrics = [ 'accuracy'])


      train_images = train_images.reshape ((TRAIN_COUNT, 28 * 28)) train_images
= train_images.astype ( 'float32') / 255

      test_images = test_images.reshape ((len (images) - TRAIN_COUNT, 28 *
      28))
      test_images = test_images.astype ( 'float32') / 255
      train_labels = to_categorical (train_labels)
      test_labels = to_categorical (test_labels)

      network.fit (train_images, train_labels, epochs = 5, batch_size = 128)

      test_acc  =  network.evaluate  (test_images,  test_labels)  [1]  print  (
"Accuracy:", test_acc)

      def main ():
      create_dataset ()
      create_network ()

      if __name__ == '__main__':

      main ()
```