Міністерство освіти і науки України  
НТУУ «КПІ ім. І. Сікорського»

Кафедра автоматизації проектування енергетичних процесів і систем

ЛАБОРАТОРНА РОБОТА № 2  
з дисципліни «МЕТОДОЛОГІЯ РОЗРОБКИ ІНТЕЛЕКТУАЛЬНИХ КОМП’ЮТЕРНИХ ПРОГРАМ»

Тема: «Розпізнавання образів за допомогою штучних нейронних мереж»

ТІ-92 Черноусов Денис

Перевірив д.т.н Мусієнко А. П.

КИЇВ 2021

**Мета роботи:**

Ознайомитися з методами розпізнавання образів за допомогою штучних нейронних мереж, побудувати, навчити та протестувати нейрону мережу для розпізнавання фігур.

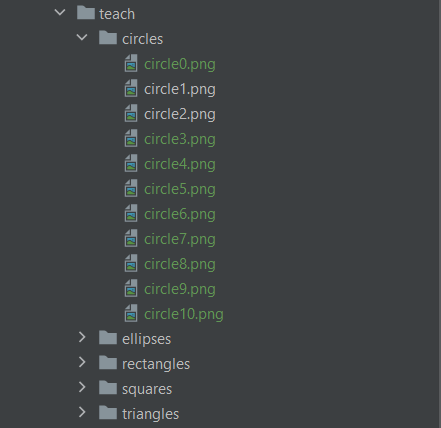
**Завдання:**

Побудувати нейронну мережу для розпізнавання образів вказаних у Вашому варіанті. Спробувати різну кількість нейронів у прихованому шарі, різну кількість прихованих шарів (без прихованого шару, з декількома прихованими шарами) та різні параметри нейронів (функція активації, алгоритм навчання, функція похибки тощо). Зробити висновки про вплив прихованих шарів та параметрів нейронів на якість розпізнавання образів. Навести хід роботи та скріншоти.

**Варіант 2**. Образи для розпізнавання - зображення кола, квадрата, еліпса, трикутника, прямокутника.

**Хід роботи**

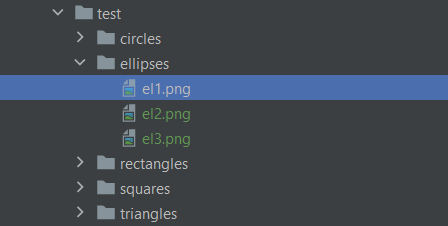
1. **Створення та робота з даними**

Створюємо папку для навчання нейронної мережі. В папці teach створюємо відповідно класам варіанту папки(circle - це кола, ellipse - еліпс)

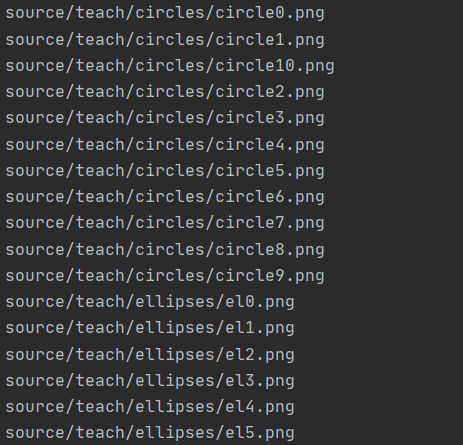
Папка teach з підкаталогами, в яких фото відповідних фігур

Папки заповнюємо відповідними фігурами різних розмірів зроблених у фотошопі. Усі фотографії надходять у розмірі 200х200 пікселів, під однаковим кутом, в центрі фото.

Зображення круга з датасету



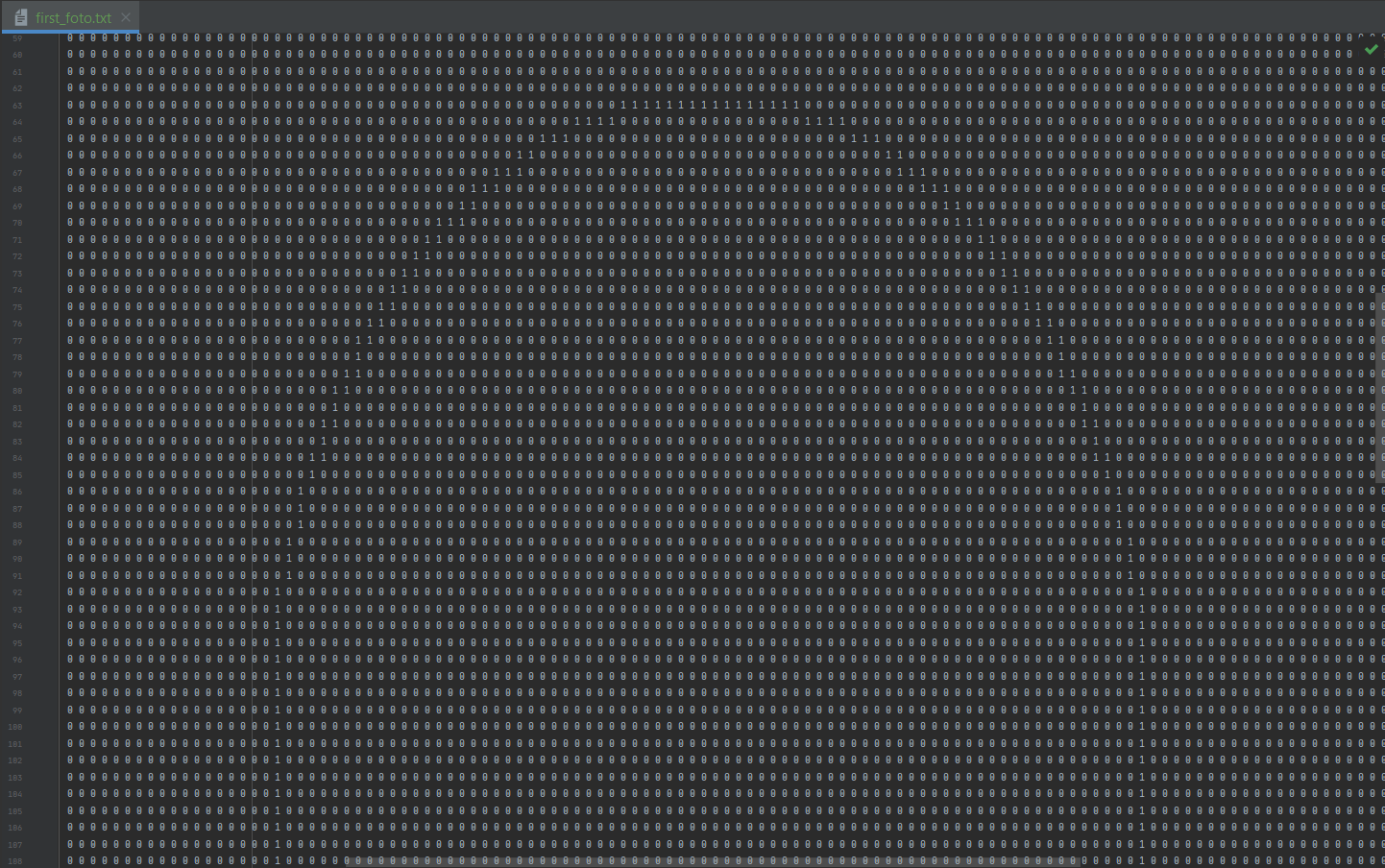
Аналогічно створюємо тестові дані

Витягаємо зображення з каталогів за допомогою функції get\_edges\_array(db) в файлі manage\_source.py. Дізнаємося відповідні класи для зображень за допомогою get\_class\_array(db) в файлі manage\_classes.py.

Обхід фото

З отриманих зображень дістаємо контури усіх фігур за допомогою бібліотеки cv2 (файл eyes.py).

Зображення контуру кола

Відповідно це зображення перетворюємо на двовимірний масив, де чорний колір це 0, а білий це 1.

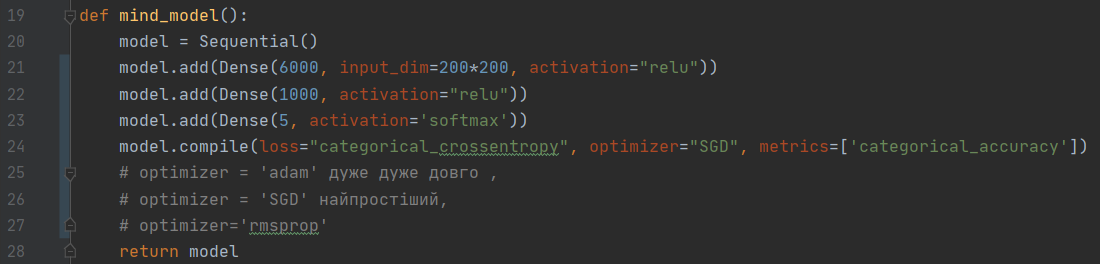
Масив чисел, що описує контур кола

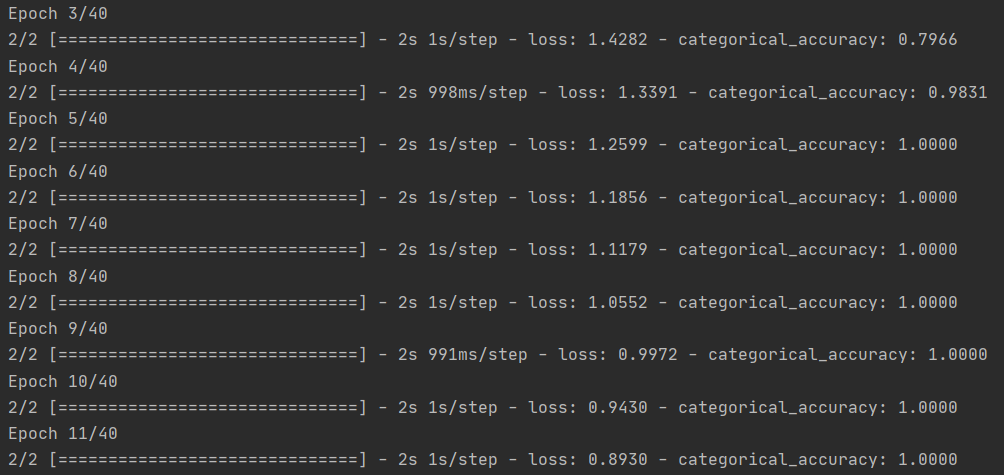
Тож, в результаті маємо масив з контурами фігур та відповідні класи для навчання нейроної мережі.

1. **Нейрона мережа**

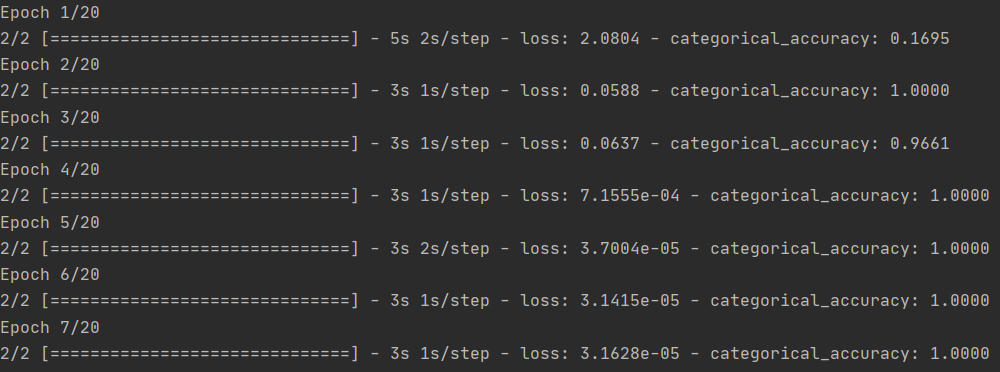
Найочивидніше, що можна було зробити це відобразити на вхід моделі 1 піксель на 1 вхід, тобто для зображення 200х200 пікселів маємо створити нейромережу з

200 \* 200 = 40 000 входів.

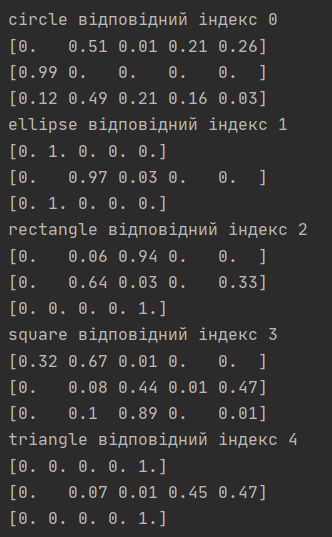


Ця версія нейронки неефективна, оскільки метод SGD є найпримітивнішую реалізацією методу градієнтного спуску, в якому навіть не передбачено накопичення імпульсу (momentum). SGD потребує великої кількості епох для проходження «ярів» і отримання точних даних, що можна побачити на скріні нище.

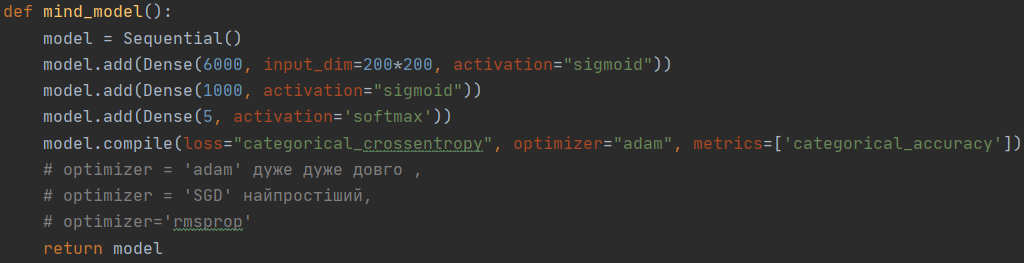
Повільний спуск з оптимізатором SGD. (див. loss - втрати)

В свою чергу, оптимізатор adam керується накопичення імпульсу (momentum).

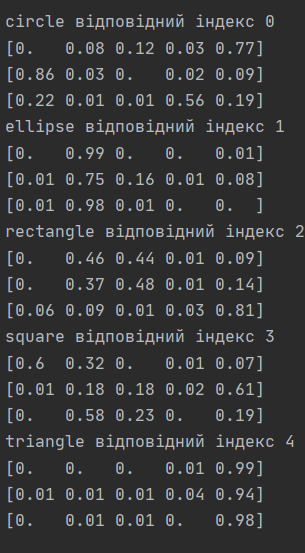
Класи тренувальних даних визначаються з 100% точністю. Похибка різко йде до нуля

Дані для тестування

Слід замітити, що система вгадала усі три фотографії трикутників, еліпсів та по одному фото прямокутника та кола.

Спробуємо змінити активаційні функції у моделі для отримання точнішого результату. Активаційна функція на виході softmax є необхідною для нашої моделі, оскілки дозволяє знайти один клас, до якого з найбільшою вірогідністю належить об’єкт (Multiclass classification). Очевидно, що сума усіх ймовірностей в такому випадку дорівнює 1.  
Для пошуку декількох класів, до яких належить об’єкт, або як ще називають визначення ознак об’єкту(Multilabel classification), використовують sigmoid функцію. Також її використовують для бінарної класифікації(Binary classification), тобто для визначення, чи належить об’єкт до класу А, чи не належить до класу А.

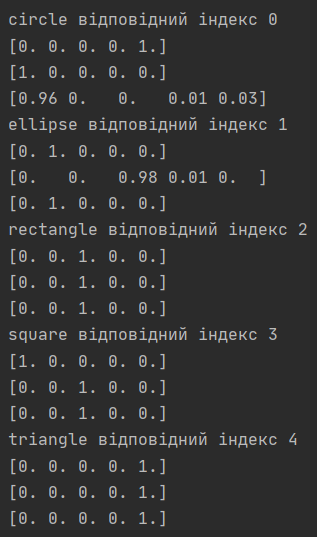
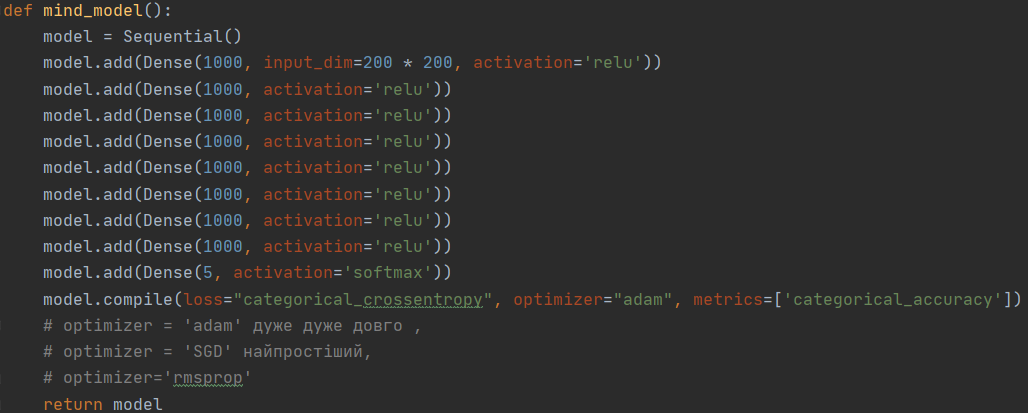
Встановлює активаційну функцію sigmoid

 Результат ідентичний до попереднього, проте кількість епох в 5 разів більша за ReLU для знаходження даних в цій точності. При такій самій кількості епох точність неймовірно низька.

При випробувані інших активаційних функцій : softmax, tanh, elu – результати не кращі ніж у ReLU, але теж потребують немалу кількість епох. Відносно непогано проявили себе linear та elu. Тож краще використовувати лінійну та кусково-лінійну активаційні функції для реалізації цієї моделі.

Спробуємо забрати прихований шар взагалі

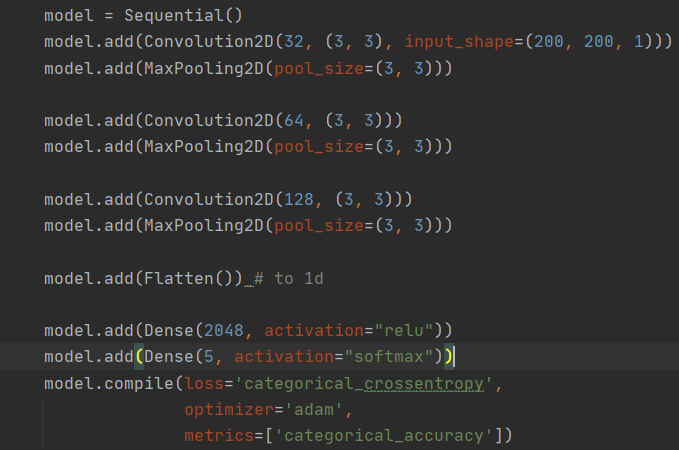
Результат очікувано жахливий, точність ніяка. Для створення мінімальної точності моделі потребується принаймні 10 тисяч епох.

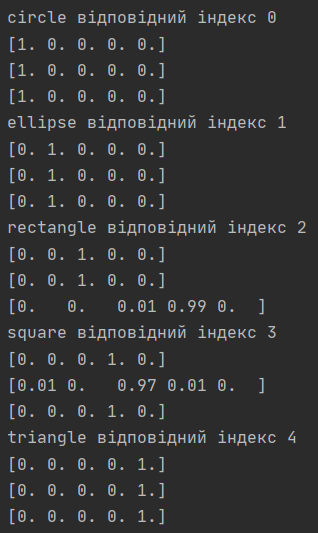
Попробуємо використати безліч прихованих шарів

Поки-що цей результат найбільш точний. Отож кількість прихованих шарів прямо таки впливає на точність вгадування системою, проте потребує 4 рази більше епох.

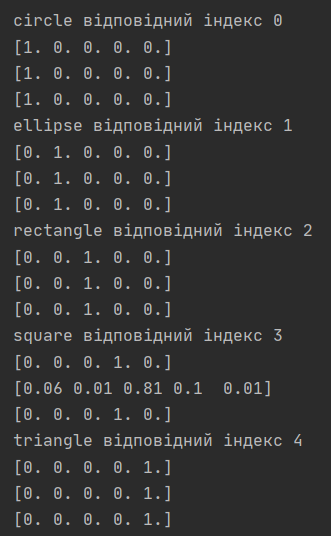
Проблема даної мережі полягає в тому, що вона нездатна розпізнати квадрат (square). Тож, побудуємо згорткову нейромережу (Convolutional Neuron Network), при цьому зберігаючи найкращі властивості архітектури попередньої.

1. **Згорткова нейрона мережа (CNN)**

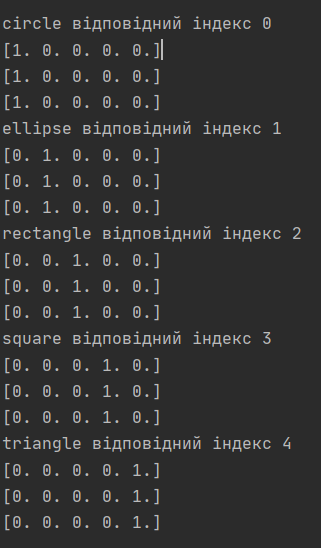
CNN



Нейромережа набагато точніша за свого попередника, проте ця система може плутати прямокутники та квадрати. Спробуємо усунути цю проблему.

Нова архітектура CNN

Система не може ідентифікувати маленький квадрат – збільшимо кількість даних про маленькі квадрати та по експериментуємо зі параметрами згортки

Ідеально.

**Код програми**

Main.py

from fibonachi\_keras.fibonachi import fibonachi\_sequence  
from lab1\_keras.lab1 import number\_sequence  
from ver1 import ver1  
from ver2 import ver2  
  
  
  
# fibonachi\_sequence()  
#  
# number\_sequence()  
#  
# ver1()  
  
ver2()

eyes.py

import cv2.cv2 as cv2  
  
  
def see(image):  
 raw\_image = cv2.imread(image, 0)  
  
 edge\_detected = cv2.Canny(raw\_image, 0, 0)  
 # cv2.imshow('Edge', edge\_detected)  
 # cv2.waitKey(0)  
  
 return edge\_detected / 255

manage\_class.py

import tensorflow as tf  
import os  
  
  
def get\_class\_array(db):  
 classes = ['circle', 'ellipse', 'rectangle', 'square', 'triangle']  
 classes\_int = []  
  
 k = 0  
  
 dirs = os.listdir(db)  
 for dir in dirs:  
 files = os.listdir('%s/%s' % (db, dir))  
 # circle, ellipses, rectangle, squares, triangles  
 for i in range(len(files)):  
 classes\_int.append(k)  
 # next type  
 k = k + 1  
  
 classes\_int = tf.keras.utils.to\_categorical(classes\_int, num\_classes=len(classes))  
  
 return classes\_int, classes

manage\_source.py

import os  
import numpy as np  
  
from eyes import see  
  
  
def get\_edges\_array(db):  
 edges = []  
  
 dirs = os.listdir(db)  
 for dir in dirs:  
 files = os.listdir('%s/%s' % (db, dir))  
 for f in files:  
 path = '%s/%s/%s' % (db, dir, f)  
 edges.append(see(path))  
 print(path)  
  
 edges = np.array(edges)  
 return edges

ver1.py

from manage\_classes import get\_class\_array  
from manage\_source import get\_edges\_array  
from first\_mind import mind\_model  
import numpy as np  
  
  
def ver1():  
 edges\_teach = get\_edges\_array('source/teach')  
  
 np.savetxt('first\_foto.txt', edges\_teach[0], fmt='%1.d')  
  
 # to 1D => size=40000  
 edges\_teach = edges\_teach.reshape(-1, 200 \* 200)  
  
 model = mind\_model()  
  
 classes\_teach\_int, classes = get\_class\_array('source/teach')  
  
 model.fit(edges\_teach, classes\_teach\_int, epochs=40)  
  
 edges\_test = get\_edges\_array('source/test')  
  
 # to 1D => size=40000  
 edges\_test = edges\_test.reshape(-1, 200 \* 200)  
  
 prs\_teach = model.predict(edges\_teach)  
 prs\_test = model.predict(edges\_test)  
  
 np.set\_printoptions(suppress=True)  
 np.set\_printoptions(precision=2)  
  
 print(prs\_teach)  
 print('-' \* 30)  
 for i in range(3 \* len(classes)):  
 if i % 3 == 0: print(classes[i // 3], "відповідний індекс", i // 3)  
 print(prs\_test[i])

first\_mind.py

from keras.models import Sequential  
from keras.layers import Convolution2D, MaxPooling2D, Dropout, Flatten, Dense, Reshape, LSTM, BatchNormalization, \  
 Convolution1D  
  
import os  
  
os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '2'  
classes = ['circle', 'ellipse', 'rectangle', 'square', 'triangle']  
  
  
def mind\_model():  
 model = Sequential()  
 model.add(Dense(4000, input\_dim=200 \* 200, activation='relu'))  
 model.add(Dense(1000, activation='relu'))  
 model.add(Dense(1000, activation='relu'))  
 model.add(Dense(1000, activation='relu'))  
 model.add(Dense(1000, activation='relu'))  
 model.add(Dense(1000, activation='relu'))  
 model.add(Dense(1000, activation='relu'))  
 model.add(Dense(1000, activation='relu'))  
 model.add(Dense(5, activation='softmax'))  
 model.compile(loss="categorical\_crossentropy", optimizer="adam", metrics=['categorical\_accuracy'])  
 # optimizer = 'adam' дуже дуже довго ,  
 # optimizer = 'SGD' найпростіший,  
 # optimizer='rmsprop'  
 return model

ver2.py

from manage\_classes import get\_class\_array  
from manage\_source import get\_edges\_array  
from second\_mind import mind\_model  
import numpy as np  
  
def ver2():  
 edges\_teach = get\_edges\_array('source/teach')  
  
 np.savetxt('first\_foto.txt', edges\_teach[0], fmt='%1.d')  
  
 # 4d => num\_foto, width, height, num\_channel  
 edges\_teach = edges\_teach.reshape(edges\_teach.shape[0], 200, 200, 1)  
  
 classes\_int\_teach, classes = get\_class\_array('source/teach')  
  
 model = mind\_model()  
  
 model.fit(edges\_teach, classes\_int\_teach, epochs=40)  
  
 edges\_test = get\_edges\_array('source/test')  
 edges\_test = edges\_test.reshape(edges\_test.shape[0], 200, 200, 1)  
  
 prs\_teach = model.predict(edges\_teach)  
 prs\_test = model.predict(edges\_test)  
  
 np.set\_printoptions(suppress=True)  
 np.set\_printoptions(precision=2)  
  
 print(prs\_teach)  
 print('-' \* 30)  
 for i in range(3 \* len(classes)):  
 if i % 3 == 0: print(classes[i // 3], "відповідний індекс", i // 3)  
 print(prs\_test[i])

second**\_**mind.py

from keras.models import Sequential  
from keras.layers import Convolution2D, MaxPooling2D, Dropout, Flatten, Dense, Reshape, LSTM, BatchNormalization, \  
 Convolution1D  
import os  
  
os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '2'  
classes = ['circle', 'ellipse', 'rectangle', 'square', 'triangle']  
  
def mind\_model():  
 model = Sequential()  
  
 model.add(Convolution2D(32, (3, 3), input\_shape=(200, 200, 1), activation="relu"))  
 model.add(MaxPooling2D(pool\_size=(2, 2)))  
  
 model.add(Convolution2D(64, (3, 3), activation="relu"))  
 model.add(MaxPooling2D(pool\_size=(2, 2)))  
  
 model.add(Convolution2D(64, (3, 3), activation="relu"))  
 model.add(MaxPooling2D(pool\_size=(3, 3)))  
  
 model.add(Dropout(0.25)) # random 1/4 of neurons are ignored each epoch  
 model.add(Flatten()) # to 1d  
  
 model.add(Dense(256, activation="relu"))  
 model.add(Dense(256, activation="relu"))  
 model.add(Dense(256, activation="relu"))  
 model.add(Dense(256, activation="relu"))  
 model.add(Dense(256, activation="relu"))  
  
 model.add(Dense(5, activation="softmax"))  
 model.compile(loss='categorical\_crossentropy',  
 optimizer='adam',  
 metrics=['categorical\_accuracy'])  
  
 # optimizer = 'adam' best for CNN,  
 # optimizer = 'adamGrad',  
 # optimizer = 'SGD',  
 # optimizer = 'adadelta'  
 return model

**Висновки**

У ході виконання лабораторної роботи було розроблено 2 нейроні мережі для розпізнання геометричних фігур, таких як коло, еліпс, прямокутник, квадрат і трикутник. Були вивчені різноманітні оптимізатори нейронних мереж, загальні принципи їх реалізації та виявлено найкращий для подібної роботи з зображеннями – adam – одна з найкращих реалізацій методу градієнтного спуску. Виявлено залежність між кількістю нейронів, кількістю прихованих шарів та точністю нейромережі. Емпірично доведено, що лінійні та кусково лінійні активаційні функції найкраще підходять для даних моделей. Збудована згорткова нейрона мережа, що набагато точніше визначає клас об’єкту та вивчено різноманітні функції для роботи з мережами даного типу. Досліджено особливості активаційних функції виводу : sigmoid, softmax, linear.