Київський національний університет
імені Т.Шевченка

Звіт

до лабораторної роботи 6
з предмету Нейронні мережі та нейрообчислення
«Класифікація датасету чисел MNIST з допомогою CNN»

Студента четвертого курсу
Групи ТК-41
Факультету комп'ютерних наук
та кібернетики
Некряча Владислава

Київ
2023

Постановка задачі

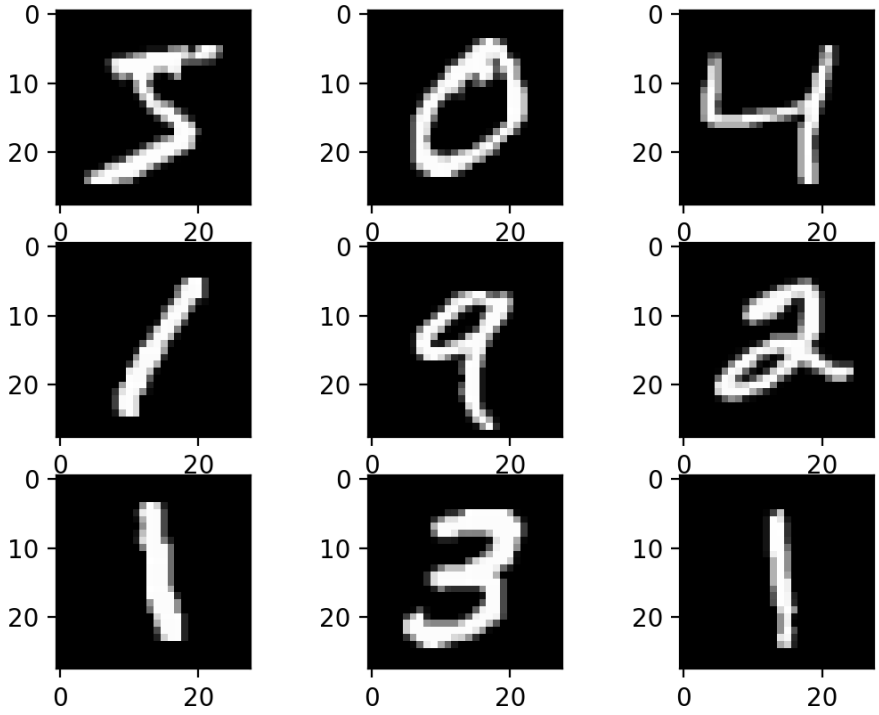
Побудувати класифікатор за допомогою CNN, який буде розпізнавати цифри з датасету MNIST.

Датасет MNIST

Датасет MNIST - це збірка зображень написаних цифр, які використовуються для тренування та тестування систем обробки зображень та алгоритмів машинного навчання. MNIST означає Modified National Institute of Standards and Technology, організацію, яка початково створила цей датасет.
Датасет складається з 70 000 зображень написаних цифр від 0 до 9, розбитих на 60 000 зображень для тренування та 10 000 зображень для тестування. Кожне зображення є зображенням у відтінках сірого кольору розміром 28x28 пікселів, кожен піксель представлений значенням від 0 до 255, що вказує на його інтенсивність.

Датасет MNIST широко використовується як бенчмарк для різних завдань машинного навчання та комп'ютерного зору, зокрема для класифікації зображень та розпізнавання цифр. Він використовується для тренування та тестування різних алгоритмів, включаючи традиційні алгоритми машинного навчання, такі як k-Nearest Neighbors та Support Vector Machines, а також глибинні моделі навчання, такі як згорткові нейронні мережі (CNNs).

Через широке використання та доступність, датасет MNIST став стандартним датасетом для порівняння продуктивності різних алгоритмів машинного навчання та комп'ютерного зору.



Згорткові мережі

Згорткові нейронні мережі (Convolutional Neural Networks, CNN) - це тип моделі глибинного навчання машинного навчання, яка зазвичай використовується для обробки зображень та інших типів даних, які можна подати у вигляді матриці.

У CNN зазвичай використовуються три типи шарів: згорткові шари, шари пулінгу та звичайні шари. Згорткові шари виконують згортку зображення з ядром (фільтром), яке складається з ваг, щоб винести високоінформативні функції зображення. Після згортки використовують шар пулінгу для зменшення розміру виходу та підвищення устійчивості до зміщення та інших змін.

Один з головних принципів, на яких базується CNN - це локальність, яка означає, що кожен невеликий фрагмент зображення може бути аналізований окремо, незалежно від інших фрагментів зображення. Це дозволяє мережі автоматично вибирати та виокремлювати характеристики зображення, такі як форми, текстури та шаблони.

CNN широко використовується для різних завдань обробки зображень, таких як класифікація зображень, детекція об'єктів, сегментація зображень та ін. Він також знайшов застосування в інших областях, таких як обробка звуку та обробка тексту.

Проте важливо зазначити, що глибокі нейронні мережі, такі як CNN, вимагають великої кількості даних та обчислювальних ресурсів для тренування та використання, і вони можуть бути вразливі до перенавчання та підгонки під дані. Тому важливо добре налаштувати параметри моделі та використовувати додаткові техніки, такі як зменшення шуму та регуляризація, щоб запобігти цим проблемам. Також, для покращення результатів, часто використовуються попередньо навчені моделі, які можна доопрацювати під свої потреби.

Практична частина:

Імпортуємо бібліотеки:

```
from numpy import mean
from numpy import std
from matplotlib import pyplot as plt
from sklearn.model_selection import KFold
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.optimizers import SGD
```

Завантажуємо датасет та використовуємо one-hot-encdoing для кодування змінної-відгуку:

```
def load_dataset():
	# load dataset
	(trainX, trainY), (testX, testY) = mnist.load_data()
	# reshape dataset to have a single channel
	trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
	testX = testX.reshape((testX.shape[0], 28, 28, 1))
	# one hot encode target values
	trainY = to_categorical(trainY)
	testY = to_categorical(testY)
	return trainX, trainY, testX, testY
```

Нормалізуємо дані:

```
def prep_pixels(train, test):
	# convert from integers to floats
	train_norm = train.astype('float32')
	test_norm = test.astype('float32')
	# normalize to range 0-1
	train_norm = train_norm / 255.0
	test_norm = test_norm / 255.0
	# return normalized images
	return train_norm, test_norm
```

Визначаємо архітектуру моделі.

Перший шар - це згортковий шар з 32 фільтрами розміром 3x3, який застосовує математичну операцію до кожної малої частини вхідного зображення для створення набору відфільтрованих зображень. Функція активації, використовувана в цьому шарі - це функція ReLU, яка зроблює вихідний сигнал позитивним і вводить нелінійність. Ініціалізатор ядра, використовуваний в цьому шарі, - це He uniform initializer, який встановлює початкові ваги фільтрів таким чином, щоб сприяти тренуванню моделі.
Другий шар - це шар максимального пулінгу, який зменшує розмір відфільтрованих зображень, беручи максимальне значення кожного маленького регіону на зображенні.

Третій шар перетворює вихід 2D з попереднього шару в 1D масив.

Четвертий шар - це повністю зв'язний шар з 100 нейронами, який застосовує функцію активації до вхідних параметрів, щоб додатково вилучити ознаки з нього. Функція активації, використовувана в цьому шарі - це ReLU, а ініціалізатор ядра - He uniform.

П'ятий шар - це повністю зв'язний шар з 10 нейронами, який використовує функцію активації softmax для передбачення ймовірності належності вхідного зображення до кожного з 10 класів (цифр від 0 до 9).

Модель компілюється з оптимізатором стохастичного градієнтного спуску (SGD) з швидкістю навчання 0,01 та коефіцієнтом моменту 0,9. Функція втрат, використовувана для підрахунку помилки - категоріальна кросс-ентропія. Метрика точності використовується для оцінки результатів моделі під час навчання і тестування.

```
# define cnn model
def define_model():
	model = Sequential()
	model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))
	model.add(MaxPooling2D((2, 2)))
	model.add(Flatten())
	model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
	model.add(Dense(10, activation='softmax'))
	# compile model
	opt = SGD(learning_rate=0.01, momentum=0.9)
	model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
	return model
```

Оцінюємо модель з допомогою кросс-валідації на 5 фолдів. Модель тренується 10 епох з перевіркою на даних для валідації, та з розміром батчу 32. Отриману модель перевіряємо на тестових даних і виводимо в консоль accuracy score.

```
# evaluate a model using k-fold cross-validation
def evaluate_model(dataX, dataY, n_folds=5):
	scores, histories = list(), list()
	# prepare cross validation
	kfold = KFold(n_folds, shuffle=True, random_state=1)
	# enumerate splits
	for train_ix, test_ix in kfold.split(dataX):
		# define model
```

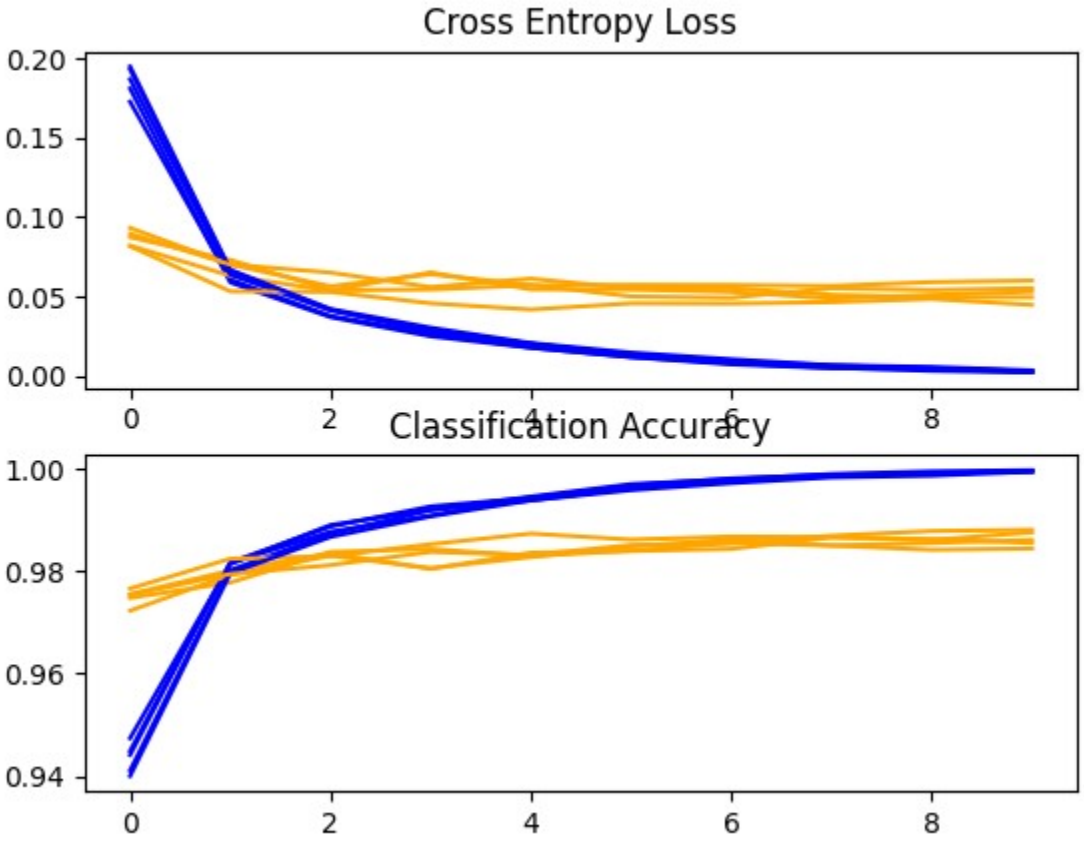
```
            model = define_model()
            # select rows for train and test
            trainX, trainY, testX, testY = dataX[train_ix], dataY[train_ix],
dataX[test_ix], dataY[test_ix]
            # fit model
            history = model.fit(trainX, trainY, epochs=10, batch_size=32,
validation_data=(testX, testY), verbose=0)
            # evaluate model
            _, acc = model.evaluate(testX, testY, verbose=0)
            print('> %.3f' % (acc * 100.0))
            # stores scores
            scores.append(acc)
            histories.append(history)
        return scores, histories
```

Результати точності на кожному з етапів кросс-валідації:

```
> 98.450
> 98.558
> 98.617
> 98.767
> 98.817
Accuracy: mean=98.642 std=0.135, n=5
```

Цей результат каже про гарну спроможність даної моделі класифікувати цифри.

Графіки точності та помилки для кожної з епох (синій – тренування, жовтий - валідація):



Box-plot точності для кросс-валідації: