МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Искусственные нейронные сети»

Тема: Распознавание рукописных символов

Студент гр. 8383 _____ Бессуднов Г. И.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2021

Цель работы

Реализовать классификацию черно-белых изображений рукописных цифр (28x28) по 10 категориям (от 0 до 9).

Основные теоретические положения

Набор данных MNIST уже входит в состав Keras в форме набора из четырех массивов Numpy.

Листинг 1:

```
import tensorflow as tf

mnist = tf.keras.datasets.mnist
(train_images, train_labels),(test_images, test_labels) = mnist.load_data()
```

Здесь train_images и train_labels — это тренировочный набор, то есть данные, необходимые для обучения. После обучения модель будет проверяться тестовым (или контрольным) набором, test_images и test_labels. Изображения хранятся в массивах Numpy, а метки — в массиве цифр от 0 до 9. Изображения и метки находятся в прямом соответствии, один к одному.

Для проверки корректности загрузки достаточно сравнить тестовое изображение с его меткой.

Листинг 2:

```
import matplotlib.pyplot as plt

plt.imshow(train_images[0],cmap=plt.cm.binary)
plt.show()
print(train_labels[0])
```

Исходные изображения представлены в виде массивов чисел в интервале [0, 255]. Перед обучением их необходимо преобразовать так, чтобы все значения оказались в интервале [0, 1].

Листинг 3:

```
train_images = train_images / 255.0
test_images = test_images / 255.0
```

Также необходимо закодировать метки категорий. В данном случае прямое кодирование меток заключается в конструировании вектора с нулевыми элементами со значением 1 в элементе, индекс которого соответствует индексу метки.

Листинг 4:

```
from keras.utils import to_categorical

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

Теперь можно задать базовую архитектуру сети.

Листинг 5:

```
from tensorflow.keras.layers import Dense, Activation, Flatten
from tensorflow.keras.models import Sequential

model = Sequential()
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

Чтобы подготовить сеть к обучению, нужно настроить еще три параметра для этапа компиляции:

1. функцию потерь, которая определяет, как сеть должна оценивать качество своей работы на обучающих данных и, соответственно, как корректировать ее в правильном направлении;
2. оптимизатор — механизм, с помощью которого сеть будет обновлять себя, опираясь на наблюдаемые данные и функцию потерь;
3. метрики для мониторинга на этапах обучения и тестирования — здесь нас будет интересовать только точность (доля правильно классифицированных изображений).

Листинг 6:

```
model.compile(optimizer='adam',loss='categorical_crossentropy',
metrics=['accuracy'])
```

Теперь можно начинать обучение сети, для чего в случае использования библиотеки Keras достаточно вызвать метод fit сети — он пытается адаптировать (fit) модель под обучающие данные.

Листинг 7:

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

В процессе обучения отображаются две величины: потери сети на обучающих данных и точность сети на обучающих данных.
Теперь проверим, как модель распознает контрольный набор:

Листинг 8:

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test_acc:', test_acc)
```

Выполнение работы

По теоретическим сведениям была построена модель и проведена ее тренировка. Код программы представлен в Приложении А. Стандартные настройки оптимизатора `Adam` дали точность в 97,86%, что уже является очень хорошим результатом и выше 95%.

Далее были проведены тренировки с другими оптимизаторами. Настройки оптимизатора выводились при помощи метода get_config(). Результаты получились следующие:

1. `SGD`

   Для этого оптимизатора было проведено два эксперимента с следующими параметрами:

   `{'name': 'SGD', 'learning_rate': 0.01, 'decay': 1e-06, 'momentum': 0.9, 'nesterov': True}.`

   Точность составила 96%.

   `{'name': 'SGD', 'learning_rate': 0.1, 'decay': 1e-06, 'momentum': 0.99, 'nesterov': True}.`

   Точность составила 90,11%.

Данный оптимизатор показывает достаточно хорошую точность при настройках первого варианта, но она все равно меньше, чем стандартный Adam.

2. RMSprop

Для этого оптимизатора было проведено несколько эксперимента с следующими параметрами:

{'name': 'RMSprop', 'learning_rate': 0.01, 'decay': 0.0, 'rho': 0.9, 'momentum': 0.0, 'epsilon': 1e-07, 'centered': False}

Точность составила 97,55%

{'name': 'RMSprop', 'learning_rate': 0.001, 'decay': 0.0, 'rho': 0.9, 'momentum': 0.0, 'epsilon': 1e-07, 'centered': False}

Точность составила 97,87%

{'name': 'RMSprop', 'learning_rate': 0.001, 'decay': 0.0, 'rho': 0.99, 'momentum': 0.0, 'epsilon': 1e-07, 'centered': False}

Точность составила 97,89%

{'name': 'RMSprop', 'learning_rate': 0.001, 'decay': 0.0, 'rho': 0.79, 'momentum': 0.0, 'epsilon': 1e-07, 'centered': False}

Точность составила 97,47%

{'name': 'RMSprop', 'learning_rate': 0.001, 'decay': 0.0, 'rho': 0.1, 'momentum': 0.0, 'epsilon': 1e-07, 'centered': False}

Точность составила 97,17%

Оптимизатор показал во всех случаях очень высокую точность, которая очень близка к стандартным настройкам Adam.

3. Adagrad

Для этого оптимизатора было проведено несколько эксперимента с следующими параметрами:

{'name': 'Adagrad', 'learning_rate': 0.01, 'decay': 0.0, 'initial_accumulator_value': 0.1, 'epsilon': 1e-07}

Точность составила 93,49%

{'name': 'Adagrad', 'learning_rate': 0.5, 'decay': 0.0, 'initial_accumulator_value': 0.1, 'epsilon': 1e-07}

Точность составила 96,4%

{'name': 'Adagrad', 'learning_rate': 0.9, 'decay': 0.0, 'initial_accumulator_value': 0.1, 'epsilon': 1e-07}

Точность составила 84,22%

Результаты данного оптимизатор не особо точны, только второй вариант настроек показал точность выше 95%, в остальных случаях, точность была достаточно низкой.

4. Adadelta

Для этого оптимизатора было проведено несколько эксперимента с следующими параметрами:

{'name': 'Adadelta', 'learning_rate': 1.0, 'decay': 0.0, 'rho': 0.95, 'epsilon': 1e-07}

Точность составила 97,63%

{'name': 'Adadelta', 'learning_rate': 0.1, 'decay': 0.0, 'rho': 0.5, 'epsilon': 1e-07}

Точность составила 92,72%

{'name': 'Adadelta', 'learning_rate': 0.5, 'decay': 0.0, 'rho': 0.5, 'epsilon': 1e-07}

Точность составила 95,65%

Оптимизатор показал неплохие результаты в первом варианте настроек (стандартные), в других вариантах результаты были хуже.

5. Adam

Для этого оптимизатора было проведено несколько эксперимента с следующими параметрами:

{'name': 'Adam', 'learning_rate': 0.1, 'decay': 0.0, 'beta_1': 0.999, 'beta_2': 0.899, 'epsilon': 1e-07, 'amsgrad': True}

Точность составила 90,88%

{'name': 'Adam', 'learning_rate': 0.1, 'decay': 0.0, 'beta_1': 0.999, 'beta_2': 0.9, 'epsilon': 1e-07, 'amsgrad': True}

Точность составила 86,79%

{'name': 'Adam', 'learning_rate': 0.01, 'decay': 0.0, 'beta_1': 0.9, 'beta_2': 0.9, 'epsilon': 1e-07, 'amsgrad': True}

Точность составила 97,91%

{'name': 'Adam', 'learning_rate': 0.01, 'decay': 0.0, 'beta_1': 0.9, 'beta_2': 0.95, 'epsilon': 1e-07, 'amsgrad': False}

Точность составила 97,14%

{'name': 'Adam', 'learning_rate': 0.5, 'decay': 0.0, 'beta_1': 0.9, 'beta_2': 0.9, 'epsilon': 1e-07, 'amsgrad': False}

Точность составила 16,98%

Данный оптимизатор использовался в изначальной настройки сети со стандартными параметрами. 3 и 4 варианты настроек смогли показать результат близкий или выше стандартных настроек. Остальные же варианты были неудачными, в особенности последний.

6. Nadam

Для этого оптимизатора было проведено несколько эксперимента с следующими параметрами:

{'name': 'Nadam', 'learning_rate': 0.002, 'decay': 0.004, 'beta_1': 0.9, 'beta_2': 0.999, 'epsilon': 1e-07}

Точность составила 97,69%

{'name': 'Nadam', 'learning_rate': 0.01, 'decay': 0.004, 'beta_1': 0.9, 'beta_2': 0.95, 'epsilon': 1e-07}

Точность составила 97,2%

Оптимизатор показал очень хорошие результаты точности, близкие или выше стандартных настроек Adam.

Далее сеть была сохранена в .h5 файл при помощи метода save(filename). Это позволило загрузить уже натренированную модель в другой программе. В файле number_recognizer.py написана программа, которая может загружать изображения пользователя и использовать ранее созданную модель для классификации изображений. Важными методами для реализации это функционала были: load_model(filename) – метод для загрузки модели из файла и Image.open(filename) – метод для загрузки изображений. Код программы представлен в Приложении Б.

Выводы

В ходе работы была реализована классификация черно-белых изображений рукописных цифр (28x28) по 10 категориям (от 0 до 9), натренирована и сохранена модель, решающая эту задачу и создана программа, с помощью которой пользователь может проверять сеть на своих файлах. Так же были проведены эксперименты с некоторыми оптимизаторами и их настройками.

ПРИЛОЖЕНИЕ А
КОД ПРОГРАММЫ MAIN.PY

```
import tensorflow as tf
import matplotlib.pyplot as plt
from keras.utils import to_categorical
from tensorflow.keras.layers import Dense, Activation, Flatten
from tensorflow.keras.models import Sequential
from keras import optimizers

mnist = tf.keras.datasets.mnist
(train_images, train_labels),(test_images, test_labels) = mnist.load_data()

opt = optimizers.Nadam(learning_rate=0.01, beta_1=0.9, beta_2=0.95)

train_images = train_images / 255.0
test_images = test_images / 255.0

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

model = Sequential()
model.add(Flatten())
model.add(Dense(256, activation="relu"))
model.add(Dense(10, activation="softmax"))

model.compile(optimizer=opt,loss="categorical_crossentropy", metrics=["accuracy"])
model.fit(train_images, train_labels, epochs=5, batch_size=128)
test_loss, test_acc = model.evaluate(test_images, test_labels)

with open("tests.txt", "a") as tests_file:
```

```
        line = "\n\n" + str(opt.get_config()) + "\n" + "Accuracy: " +
str(test_acc)
        tests_file.write(line)


    print("test_acc:", test_acc)

    print(opt.get_config())

    model.save("model_lb4.h5")
```

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ NUMBER_RECOGNIZER.PY

```
from keras.models import load_model
from PIL import Image
import numpy as np
from pathlib import Path

def predict_all():
    for i in range(10):
        path = Path(str(i) + ".png")
        img = Image.open(path.absolute())
        img = np.array(img) / 255.0
        pred = model.predict(np.array([img]))
        print("Prediction for numder", i, ":", np.argmax(pred))

def predict_one(filename):
    path = Path(filename)
    if (not path.exists()):
        print("Oops! Can't find file", filename)
        return
    img = Image.open(path.absolute())
    img = np.array(img) / 255.0
    pred = model.predict(np.array([img]))
    print("Prediction for file", filename, ":", np.argmax(pred))



model = load_model("model_lb4.h5")
print("Predict all files?(y/n)")
inp = str(input())
while inp != "y" and inp != "n":
    print("Wrong input. Predict all files?(y/n)")
    inp = input()
```

```
if inp == "y":
    predict_all()
else:
    print("Enter file name:")
    filename = input()
    predict_one(filename)
```