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

ОТЧЕТ

по практической работе №8

по дисциплине «Искусственные нейронные сети»

Тема: Ансамблирование моделей нейронных сетей с использованием библиотеки Keras

Студентка гр. 8383 _____ Максимова А.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы

Необходимо реализовать собственный CallBack, и провести обучение собственной модели из практического занятия № 6 с написанным CallBack'ом. То, какой CallBack необходимо реализовать определяется вариантом.

Задание

1 вариант

Сохранение трех наилучших моделей. Название файлов с моделями должны иметь следующий вид <текущая дата>_<префикс, задаваемый пользователем>_<номер модели>

Выполнение работы

1. Используемая модель сверточной нейронной сети, классифицирующей черно-белые изображения по количеству крестов на них (может быть 1, 2, 3).

```
inp = Input(shape=(height, width, depth))
conv_1 = Convolution2D(filters=conv_depth_1, kernel_size=(kernel_size, kernel_size),
                       padding='same', activation='relu')(inp)
pool_1 = MaxPooling2D(pool_size=(pool_size, pool_size))(conv_1)
drop_1 = Dropout(rate=drop_prob_1)(pool_1)


conv_2 = Convolution2D(filters=conv_depth_2, kernel_size=(kernel_size, kernel_size),
                       padding='same', activation='relu')(drop_1)
pool_2 = MaxPooling2D(pool_size=(pool_size, pool_size))(conv_2)
drop_2 = Dropout(rate=drop_prob_1)(pool_2)

flat = Flatten()(drop_2)
dense_1 = Dense(dense_size_1, activation='relu')(flat)
drop_3 = Dropout(rate=drop_prob_2)(dense_1)
out = Dense(dense_size_2, activation='softmax')(drop_3)

model = Model(inputs=inp, outputs=out)
model.compile(loss='categorical_crossentropy', optimizer='Adam', metrics=['accuracy'])
```

2. Были переопределены следующие функции:

```
class CallbackVar1(Callback):  # создали подкласс Callback

    def __init__(self, prefix):
        date = datetime.now()
        self.file_name = '{}-{}-{}_{}_'.format(date.day, date.month, date.year, prefix)

        self.elem = 3
        self.val_acc = [0] * self.elem
        self.rewriting = 0
        self.param = 'val_accuracy'

    def on_epoch_end(self, epoch, logs=None):
        for i in range(self.elem):
            acc_epoch = logs[self.param]
            if self.val_acc[i] < acc_epoch:
                self.val_acc.insert(i, acc_epoch)
                self.model.save(self.file_name + str(i + 1) + '.hdf5')
                if self.rewriting != self.elem - 1:
                    self.rewriting = self.rewriting + 1
                else:
                    self.rewriting = 0
                break

            elif self.rewriting > i:
                continue

    def on_train_end(self, logs=None):
        print("Тройка наилучших моделей:")
        for i in range(self.elem):
            print(self.file_name + str(i + 1) + '.hdf5:' + " val_accuracy = " + str(self.val_acc[i]))
```

Лучшими моделями будем считать модели с наибольшей точностью на валидационных данных.

Создаем список из трех элементов, заполненный нулями.

В конце каждой эпохи проверяем нельзя ли обновить один из трех элементов списка: если значение точности сети на валидационных данных на текущей эпохе превышает одно из них. При этом одно и то же значение точности сети *с конкретной эпохи* не может быть записано в более, чем один элемент списка.

В конце тренировки выводим полученные результаты.

Значение *prefix* задается пользователем с консоли.

3. Тестирование:

Epoch 1/15
141/141 [==============================] - 5s 35ms/step - loss: 0.9065 - accuracy: 0.5203 - val_loss: 0.3242 - val_accuracy: 0.9000
Epoch 2/15
141/141 [==============================] - 4s 29ms/step - loss: 0.3025 - accuracy: 0.8921 - val_loss: 0.1604 - val_accuracy: 0.9573
Epoch 3/15
141/141 [==============================] - 4s 29ms/step - loss: 0.2048 - accuracy: 0.9165 - val_loss: 0.1214 - val_accuracy: 0.9600
Epoch 4/15
141/141 [==============================] - 4s 29ms/step - loss: 0.1447 - accuracy: 0.9564 - val_loss: 0.0785 - val_accuracy: 0.9827
Epoch 5/15
141/141 [==============================] - 4s 29ms/step - loss: 0.1287 - accuracy: 0.9530 - val_loss: 0.0819 - val_accuracy: 0.9773
Epoch 6/15
141/141 [==============================] - 4s 28ms/step - loss: 0.0917 - accuracy: 0.9748 - val_loss: 0.0816 - val_accuracy: 0.9720
Epoch 7/15
141/141 [==============================] - 4s 28ms/step - loss: 0.1203 - accuracy: 0.9544 - val_loss: 0.0657 - val_accuracy: 0.9827
Epoch 8/15
141/141 [==============================] - 4s 28ms/step - loss: 0.0746 - accuracy: 0.9752 - val_loss: 0.1236 - val_accuracy: 0.9640
Epoch 9/15
141/141 [==============================] - 4s 29ms/step - loss: 0.0695 - accuracy: 0.9760 - val_loss: 0.4668 - val_accuracy: 0.8187
Epoch 10/15
141/141 [==============================] - 4s 29ms/step - loss: 0.1190 - accuracy: 0.9574 - val_loss: 0.1209 - val_accuracy: 0.9680
Epoch 11/15
141/141 [==============================] - 4s 29ms/step - loss: 0.1452 - accuracy: 0.9400 - val_loss: 0.0767 - val_accuracy: 0.9787
Epoch 12/15
141/141 [==============================] - 4s 29ms/step - loss: 0.0694 - accuracy: 0.9791 - val_loss: 0.1010 - val_accuracy: 0.9747
Epoch 13/15
141/141 [==============================] - 4s 30ms/step - loss: 0.0535 - accuracy: 0.9811 - val_loss: 0.0680 - val_accuracy: 0.9827
Epoch 14/15
141/141 [==============================] - 4s 29ms/step - loss: 0.0430 - accuracy: 0.9807 - val_loss: 0.0923 - val_accuracy: 0.9787
Epoch 15/15
141/141 [==============================] - 4s 29ms/step - loss: 0.0454 - accuracy: 0.9840 - val_loss: 0.0576 - val_accuracy: 0.9853
Тройка наилучших моделей:
21-5-2021_model_1.hdf5: val_accuracy = 0.9853333234786987
21-5-2021_model_2.hdf5: val_accuracy = 0.9826666712760925
21-5-2021_model_3.hdf5: val_accuracy = 0.9826666712760925
32/32 [==============================] - 0s 12ms/step - loss: 0.0669 - accuracy: 0.9820