Практическое задание №6

8383 Ишанина Людмила, вариант 7

Задание:

Необходимо построить сверточную нейронную сеть, которая будет классифицировать черно-белые изображения с простыми геометрическими фигурами на них.
К каждому варианту прилагается код, который генерирует изображения.
Для генерации данных необходимо вызвать функцию gen_data, которая возвращает два тензора:

1. Тензор с изображениями ранга 3
2. Тензор с метками классов

Обратите внимание:

- Выборки не перемешаны, то есть наблюдения классов идут по порядку
- Классы характеризуются строковой меткой
- Выборка изначально не разбита на обучающую, контрольную и тестовую
- Скачивать необходимо оба файла. Подключать файл, который начинается с var (в нем и находится функция gen_data)

Реализация:

Была реализована модель, представленная в листинге ниже:

```
model = Sequential()
model.add(Conv2D(16, kernel_size=(7, 7), activation='relu', input_shape=(5
0, 50, 1), padding='same'))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Dropout(0.25))
model.add(Conv2D(32, kernel_size=(7, 7), activation='relu', padding='same'
))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Dropout(0.25))
model.add(Conv2D(64, kernel_size=(7, 7), activation='relu', padding='same'
))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(80, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(3, activation='softmax'))
```

Также была выбрана функция потерь `loss='categorical_crossentropy'`

И оптимизатор `optimizer='adam'`

В ходе тестирования выяснилось, что исходные изображения генерируются не всегда правильно, например, вместо трех линий на изображении две. Были перепробованы разные модели, но добиться точности выше 72% не получилось из-за этих ошибок во входных данных. Для того чтобы сеть хоть как-то обучалась, пришлось сделать модель довольно сложной (более простые модели выдавали точность 60%).

Примеры работы программы:

```
Epoch 1/12
180/180 [==============================] - 1s 5ms/step - loss: 1.0509 -
accuracy: 0.4174 - val_loss: 0.8613 - val_accuracy: 0.6250
Epoch 2/12
180/180 [==============================] - 1s 4ms/step - loss: 0.8331 -
accuracy: 0.5988 - val_loss: 0.7452 - val_accuracy: 0.6775
Epoch 3/12
180/180 [==============================] - 1s 4ms/step - loss: 0.7466 -
accuracy: 0.6444 - val_loss: 0.6594 - val_accuracy: 0.6925
Epoch 4/12
180/180 [==============================] - 1s 4ms/step - loss: 0.7257 -
accuracy: 0.6521 - val_loss: 0.6593 - val_accuracy: 0.6400
Epoch 5/12
180/180 [==============================] - 1s 4ms/step - loss: 0.7155 -
accuracy: 0.6582 - val_loss: 0.6207 - val_accuracy: 0.6925
Epoch 6/12
180/180 [==============================] - 1s 3ms/step - loss: 0.6819 -
accuracy: 0.6746 - val_loss: 0.6043 - val_accuracy: 0.7250
Epoch 7/12
180/180 [==============================] - 1s 4ms/step - loss: 0.6755 -
accuracy: 0.6893 - val_loss: 0.5959 - val_accuracy: 0.6925
Epoch 8/12
180/180 [==============================] - 1s 4ms/step - loss: 0.6536 -
accuracy: 0.6868 - val_loss: 0.6516 - val_accuracy: 0.7075
Epoch 9/12
180/180 [==============================] - 1s 3ms/step - loss: 0.6528 -
accuracy: 0.7060 - val_loss: 0.6295 - val_accuracy: 0.7100
Epoch 10/12
180/180 [==============================] - 1s 3ms/step - loss: 0.6346 -
accuracy: 0.7139 - val_loss: 0.5976 - val_accuracy: 0.6925
Epoch 11/12
180/180 [==============================] - 1s 3ms/step - loss: 0.6256 -
accuracy: 0.7135 - val_loss: 0.6165 - val_accuracy: 0.7150
Epoch 12/12
180/180 [==============================] - 1s 4ms/step - loss: 0.6438 -
accuracy: 0.7046 - val_loss: 0.5813 - val_accuracy: 0.7275
32/32 [==============================] - 0s 3ms/step - loss: 0.6133 -
accuracy: 0.7270
```