Шишкин Иван, гр. 8383

Вариант 3

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)

Выполнение работы.

Так как выборки не перемешаны изначально, то они были перемешаны с помощью функции shuffle. Из-за того, что классы характеризуются строковой меткой, то они преобразуются к виду, который нейронная сеть способна понять (листинг 1).

Листинг 1 – Подготовка к созданию модели

```
X, Y = var3.gen_data()
X, Y = shuffle(X, Y)
encoder = LabelEncoder()
encoder.fit(Y)
encoded_Y = encoder.transform(Y)
encoded_Y = np_utils.to_categorical(encoded_Y, 2)
```

Затем производится разбиение данных на обучающую (train), контрольную (val) и тестовую (test) с помощью функции breakdown_of_data (листинг 2). Тестовые данные занимают 25% от всех данных, контрольные – 25% от оставшихся данных, и все остальные данные – обучающие.

Листинг 2 – Функция breakdown_of_data

```
def breakdown_of_data(data, label):
    size = len(data)
    test_size = size // 4
    test_data = data[:test_size]
    test_label = label[:test_size]
    data = data[test_size:]
    label = label[test_size:]

    val_size = size // 4
    val_data = data[:val_size]
    val_label = label[:val_size]
    data = data[val_size:]
    label = label[val_size:]

    return (data[:], val_data, test_data), (label[:], val_label, test_label)
```

Затем задаются гиперпараметры сети (листинг 3).

Листинг 3 – Гиперпараметры

```
batch_size = 32
num_epochs = 10
kernel_size = 4
pool_size = 4
conv_depth_1 = 32
conv_depth_2 = 64
drop_prob_1 = 0.25
drop_prob_2 = 0.5
hidden_size = 512
```

После этого создается сама модель (листинг 4).

Листинг 4 – Создание модели

```
conv_1 = Convolution2D(conv_depth_1, (kernel_size, kernel_size), padding='same', activation='relu')(inp)
```

```
conv_2 = Convolution2D(conv_depth_1, (kernel_size, kernel_size),
padding='same', activation='relu')(conv_1)
pool_1 = MaxPooling2D(pool_size=(pool_size, pool_size))(conv_2)
drop_1 = Dropout(drop_prob_1)(pool_1)

conv_3 = Convolution2D(conv_depth_2, (kernel_size, kernel_size),
padding='same', activation='relu')(drop_1)
conv_4 = Convolution2D(conv_depth_2, (kernel_size, kernel_size),
padding='same', activation='relu')(conv_3)
pool_2 = MaxPooling2D(pool_size=(pool_size, pool_size))(conv_4)
drop_2 = Dropout(drop_prob_1)(pool_2)

flat = Flatten()(drop_2)
hidden = Dense(hidden_size, activation='relu')(flat)
drop_3 = Dropout(drop_prob_2)(hidden)
out = Dense(num_classes, activation='softmax')(drop_3)

model = Model(inputs=inp, outputs=out)  # To define a model, just specify
its input and output layers

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

model.fit(train_x, train_y,
          batch_size=batch_size, epochs=num_epochs,
          verbose=2, validation_data=(val_x, val_y))

evaluate_model = model.evaluate(test_x, test_y, verbose=2)
```

Вывод программы:

Epoch 10/10

10/10 - 1s - loss: 0.0965 - accuracy: 0.9367 - val_loss: 0.1550 - val_accuracy: 0.9200

4/4 - 0s - loss: 0.0926 - accuracy: 0.9200

Точность на обучающих данных составила 94%, на контрольных и тестовых – 92%.