Задание

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

Вариант 5

Классификация изображений с прямоугольником или не закрашенным кругом.

Подготовка данных

Подготовка данных для нейронной сети состоит из следующих этапов:

- Генерация данных
- Кодирование меток
- Перемешивание данных
- Нормирование данных
- Разбиение данных на тренировочное и тестовое множества

Данные генерируются при помощи предоставленной в задании функции `gen_data`:

```
data, labels = var5.gen_data()
```

Так как в задании необходимо осуществить бинарную классификацию, текстовые метки класса можно заменить числами 1 и 0:

```
sz = data.shape[0]
img_size = data.shape[1]
labels = labels.reshape(sz)
encoder = LabelEncoder()
encoder.fit(labels)
labels = encoder.transform(labels).reshape(sz, 1)
```

Для того, чтобы выходные данные соответствовали входным, перед перемешиванием объединяем их в один тензор, а затем перемешанный тензор делим обратно на входные и выходные данные.

```
data = data.reshape(sz, img_size**2)
data = np.hstack((data, labels))
rng = np.random.default_rng()
rng.shuffle(data)
labels = data[:, -1].reshape(sz, 1)
data = data[:, :data.shape[1]-1].reshape(sz, img_size,
img_size, 1)
```

Входные данные нормализуются: каждый элемент тензора делится на значение максимального элемента по всему тензору, таким образом абсолютные значения всех элементов будут расположены в интервале [0, 1].

```
data /= np.max(data)
```

Полученные данные делим на тренировочное и контрольное множества: в тренировочное множество попадут первые 90% данных, в контрольное – оставшиеся 10%.

```
tr_sz = int(sz*0.9 // 1)
train_data = data[:tr_sz, :]
train_labels = labels[:tr_sz, :]
test_data = data[tr_sz:, :]
test_labels = labels[tr_sz:, :]
```

Построение модели

Построим модель нейронной сети, состоящей из двух сверточных слоев, после каждого из которых следует слой субдискретизации, и двух полносвязных слоев. Зададим гиперпараметры сети:

```
kernel_size = 3
pool_size = 2
conv_depth_1 = 16
```

```
conv_depth_2 = 32
hidden_size = 100
```

В обоих сверточных слоях используется ядро свертки размера 3×3, первый сверточный слой применяет к изображению 16 фильтров, второй – 32 фильтра. Субдискретизация осуществляется по окнам размера 2×2. Первый полносвязный слой состоит из 100 нейронов и использует функцию активации *relu*. Второй полносвязный слой состоит из одного нейрона с функцией активации *sigmoid* – он распределяет изображения по классам. В качестве функции потерь будем использовать бинарную энтропию, минимизировать потери будем при помощи оптимизатора *Adam*.

```
inp = Input(shape=(img_size, img_size, 1))
conv_1 = Convolution2D(conv_depth_1, kernel_size,
padding='same', activation='relu')(inp)
pool_1 = MaxPooling2D(pool_size=(pool_size,
pool_size))(conv_1)
conv_2 = Convolution2D(conv_depth_2, kernel_size,
padding='same', activation='relu')(pool_1)
pool_2 = MaxPooling2D(pool_size=(pool_size,
pool_size))(conv_2)
flat = Flatten()(pool_2)
hidden = Dense(hidden_size, activation='relu')(flat)
out = Dense(1, activation='sigmoid')(hidden)

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

Оценка точности модели

Модель обучается в течение 12 эпох пакетами по 10 образцов. 10% входных данных используются для валидации. На рис. 1 и 2 приведены графики потерь и точности модели в процессе обучения на тренировочных и валидационных данных.

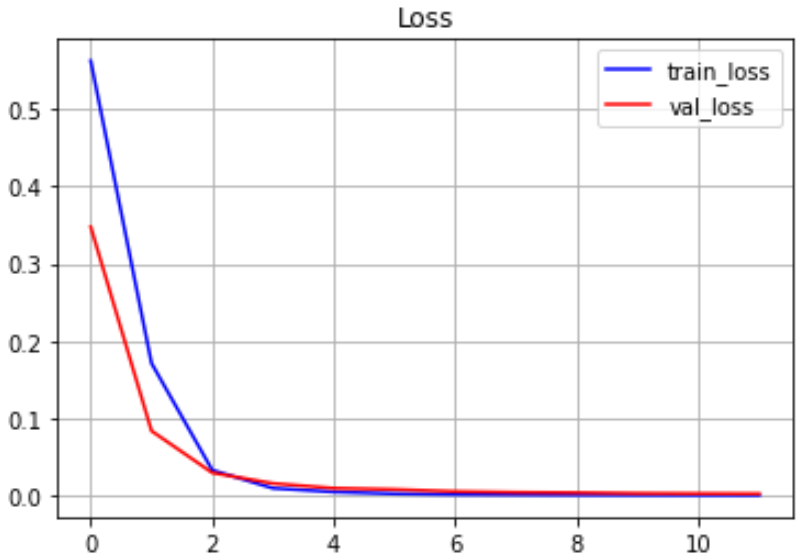

Рисунок 1

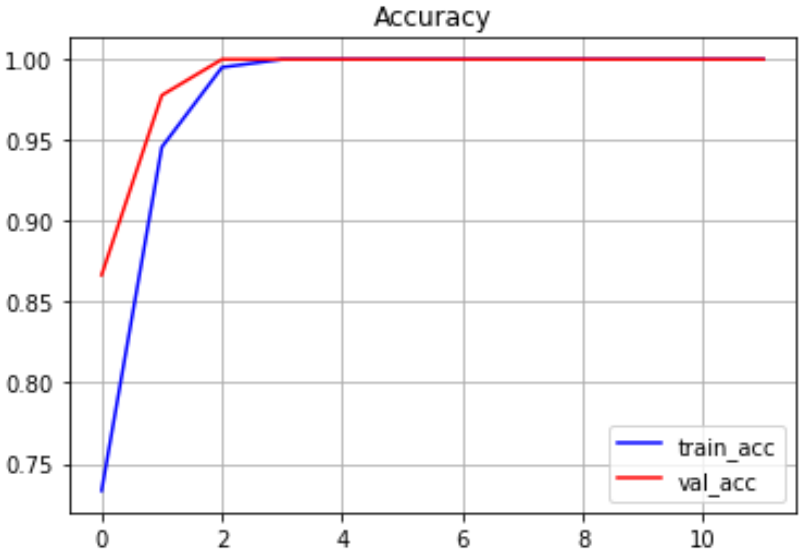

Рисунок 2

Как видно на графиках, модель достигает высокой точности на обоих множествах, причем очень быстро.

Оценим модель на тестовом множестве:

```
2/2 [==============================] - 0s 15ms/step - loss:
0.0044 - accuracy: 1.0000
```

Получили точность 100%.