Постановка задачи.

Вариант 1

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

Сначала был подключен файл генерация входных данных, сгенерированы данные для обучения модели в количестве 1000 единиц, затем данные были разбиты на тренировочные и тестовые.

```
data, label = gen_data(1000)
length = len(label)
test_data_length = length // 5

encoder = LabelEncoder()
encoder.fit(label)
labels = encoder.transform(label)

temp = list(zip(data, labels))
np.random.shuffle(temp)
data, labels = zip(*temp)
data = np.asarray(data).reshape(length, 50, 50, 1)
labels = np.asarray(labels).flatten()

train_data = data[test_data_length:length]
train_labels = labels[test_data_length:length]
```

```
test_data = data[:test_data_length]
test_labels = labels[:test_data_length]
```

Затем была создана и обучена модель нейронной сети. Модель состоит из 9 слоев, с двумя свёрточными, двух слоев субдискретизации, одного прореживающего слоя и одного входного слоя.

```
      model = Sequential()
model.add(Input(shape=(50, 50, 1)))
model.add(Conv2D(16, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, kernel_size=(2, 2), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',                    optimizer='adam',
metrics=['accuracy'])
history = model.fit(train_data, train_labels, epochs=20, batch_size=10,
              validation_split=0.2)
```