Был выполнен 2 вариант.

Функция для перемешивания данных:

```
def shuffle_data(data, label):
    tmp_data = []
    for i in range(len(data)):
        tmp_data.append([data[i], label[i]])
    random.shuffle(tmp_data)
    res_data = []
    res_label = []
    for i in range(len(data)):
        res_data.append(tmp_data[i][0])
        res_label.append(tmp_data[i][1])
    return np.asarray(res_data), np.asarray(res_label)
```

Функция для деления данных на обучающую, валидационную и тестовую выборки:

```
def split_data(data, label):
    size = len(data)
    test_size = size // 5
    train_size = size - test_size
    val_size = train_size // 5
    train_size -= val_size
    return (data[0:train_size], data[train_size:train_size + val_size],
data[train_size + val_size:]), \
           (label[0:train_size], label[train_size:train_size + val_size],
label[train_size + val_size:])
```

Генерация, перемешивание и разбиение данных:

```
x, y = generator.gen_data()
print(x.shape)
y = np.asarray([[0.] if item == 'Empty' else [1.] for item in y])
x, y = shuffle_data(x, y)
(train_x, val_x, test_x), (train_y, val_y, test_y) = split_data(x, y)
```

Параметры для построения и обучения нейросети:

```
batch_size = 32
num_epochs = 10

kernel_size = 3
pool_size = 2
conv_depth_1 = 32
conv_depth_2 = 64
drop_prob_1 = 0.25
hidden_size = 512
```

Архитектура нейросети:

```
inp = Input(shape=(50, 50, 1))

conv_1 = Convolution2D(conv_depth_1, (kernel_size, kernel_size),
padding='same', activation='relu', data_format='channels_last')(inp)
conv_2 = Convolution2D(conv_depth_1, (kernel_size, kernel_size),
padding='same', activation='relu', data_format='channels_last')(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', data_format='channels_last')(drop_1)
conv_4 = Convolution2D(conv_depth_2, (kernel_size, kernel_size),
padding='same', activation='relu', data_format='channels_last')(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_1)(hidden)
out = Dense(1, activation='sigmoid')(drop_3)
```

Обучение и тестирование модели

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

model.fit(train_x, train_y,
          batch_size=batch_size, epochs=num_epochs,
          verbose=1, validation_data=(val_x, val_y))
model.evaluate(test_x, test_y, verbose=True)
```

Результат:

```
Epoch 1/10

10/10 [==============================] - 3s 219ms/step - loss: 0.7303
- accuracy: 0.4872 - val_loss: 0.6843 - val_accuracy: 0.5750

Epoch 2/10

10/10 [==============================] - 1s 142ms/step - loss: 0.6469
- accuracy: 0.6450 - val_loss: 0.6520 - val_accuracy: 0.6125

Epoch 3/10

10/10 [==============================] - 1s 142ms/step - loss: 0.5543
- accuracy: 0.7402 - val_loss: 0.6157 - val_accuracy: 0.7125

Epoch 4/10

10/10 [==============================] - 2s 153ms/step - loss: 0.4168
- accuracy: 0.7967 - val_loss: 0.4655 - val_accuracy: 0.7250

Epoch 5/10

10/10 [==============================] - 1s 144ms/step - loss: 0.2643
- accuracy: 0.8920 - val_loss: 0.6168 - val_accuracy: 0.7625
```

```
Epoch 6/10
10/10 [==============================] - 1s 144ms/step - loss: 0.2376
- accuracy: 0.8915 - val_loss: 0.7385 - val_accuracy: 0.8000
Epoch 7/10
10/10 [==============================] - 1s 144ms/step - loss: 0.1378
- accuracy: 0.9552 - val_loss: 0.5580 - val_accuracy: 0.8500
Epoch 8/10
10/10 [==============================] - 2s 155ms/step - loss: 0.1035
- accuracy: 0.9501 - val_loss: 0.4214 - val_accuracy: 0.8750
Epoch 9/10
10/10 [==============================] - 1s 143ms/step - loss: 0.0728
- accuracy: 0.9704 - val_loss: 0.3704 - val_accuracy: 0.9250
Epoch 10/10
10/10 [==============================] - 1s 142ms/step - loss: 0.0306
- accuracy: 0.9876 - val_loss: 0.4882 - val_accuracy: 0.9500
4/4 [==============================] - 0s 20ms/step - loss: 0.1893 -
accuracy: 0.9200
```

В результате точность на тренировочных данных 98.76%, на валидационных – 95%, на тестовых – 92%.