Был выполнен 1 вариант. В качестве цели регрессии использовалась функция 7 : $\frac{x^3}{4} + e$

Генерация обучающих и тестовых данных и запись их в csv файле:

```
def gen_data(size=1000, mx=3, dx=10, me=0, de=0.3):
    size1 = size // 2
    size2 = size - size1

    train_data = gen(size1, mx, dx, me, de)
    test_data = gen(size2, mx, dx, me, de)

    return train_data, test_data


def gen(size, mx=3, dx=10, me=0, de=0.3):
    all_x = np.reshape(np.random.normal(mx, dx, size), (size, 1))
    print("X:")
    print(all_x)
    e = np.reshape(np.random.normal(me, de, size), (size, 1))
    inp = np.asarray([[f1(all_x[i], e[i]), f2(all_x[i], e[i]), f3(all_x[i],
e[i]),
                       f4(all_x[i], e[i]), f5(all_x[i], e[i]), f6(all_x[i],
e[i])] for i in range(size)])
    inp = np.reshape(inp, (size, 6))
    out = np.asarray([f7(x, np.random.normal(me, de, 1)) for x in all_x])
    return np.hstack((inp, out))


def f1(x, e):
    return x**2 + e


def f2(x, e):
    return math.sin(x/2) + e


def f3(x, e):
    return math.cos(2 * x) + e


def f4(x, e):
    return x - 3 + e


def f5(x, e):
    return -x + e


def f6(x, e):
    return math.fabs(x) + e


def f7(x, e):
    return (x**3)/4 + e


train, test = gen_data(2000)
np.savetxt("train.csv", train, delimiter=";")
np.savetxt("test.csv", test, delimiter=";")
```

Получение данных из csv:

```
train_data = np.genfromtxt('train.csv', delimiter=';')
test_data = np.genfromtxt('test.csv', delimiter=';')

train_x = np.reshape(train_data[:, :6], (len(train_data), 6))
train_y = np.reshape(train_data[:, 6], (len(train_data), 1))
test_x = np.reshape(test_data[:, :6], (len(test_data), 6))
test_y = np.reshape(test_data[:, 6], (len(test_data), 1))
```

Допустим, в качестве кодирования используется следующее преобразование:

```
train_encoded_x = np.asarray([[arr[0], arr[3], arr[4], arr[5]] for arr in
train_x])
test_encoded_x = np.asarray([[arr[0], arr[3], arr[4], arr[5]] for arr in
train_x])
```

Архитектура модели:

```
main_input = Input(shape=(6,), name='main_input')

encoding_layer = Dense(16, activation='relu')(main_input)
encoding_layer = Dense(16, activation='relu')(encoding_layer)
encoding_layer = Dense(16, activation='relu')(encoding_layer)
encoding_output = Dense(4, name='encoding_output')(encoding_layer)

decoding_layer = Dense(64, activation='relu')(encoding_output)
decoding_layer = Dense(64, activation='relu')(decoding_layer)
decoding_layer = Dense(64, activation='relu')(decoding_layer)
decoding_output = Dense(6, name='decoding_output')(decoding_layer)

regression_layer = Dense(64, activation='relu')(encoding_output)
regression_layer = Dense(64, activation='relu')(regression_layer)
regression_layer = Dense(64, activation='relu')(regression_layer)
regression_output = Dense(1, name='regression_output')(regression_layer)

model = Model(inputs=[main_input], outputs=[regression_output,
encoding_output, decoding_output])
```

Обучение модели полностью:

```
model.compile(optimizer='rmsprop', loss='mse', metrics='mae')
model.fit([train_x], [train_y, train_encoded_x, train_x], epochs=200,
batch_size=5, validation_split=0)
```

Создание тестирование и сохранение 3-ех моделей:

```
regression_model = Model(inputs=[main_input], outputs=[regression_output])
regression_prediction = regression_model.predict(test_x)

encoding_model = Model(inputs=[main_input], outputs=[encoding_output])
encoding_prediction = encoding_model.predict(test_x)

decoding_model = Model(inputs=[main_input], outputs=[decoding_output])
decoding_prediction = decoding_model.predict(test_x)
```

```
regression_model.save('regression_model.h5')
encoding_model.save('encoding_model.h5')
decoding_model.save('decoding_model.h5')

np.savetxt('regression_results.csv', np.hstack((test_y,
regression_prediction)), delimiter=';')
np.savetxt('encoding_results.csv', np.hstack((test_encoded_x,
encoding_prediction)), delimiter=';')
np.savetxt('decoding_results.csv', np.hstack((test_x, decoding_prediction)),
delimiter=';')
```