Завершение кода

In [72]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import seaborn as sns
import plotly.express as px
import tensorflow as tf
import sklearn

from sklearn import linear_model
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.linear_model import LinearRegression, LogisticRegression, SGDRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_percentage_error, mean_absolute_error
from sklearn.model_selection import train_test_split, GridSearchCV, KFold, cross_val_score
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn import preprocessing
from sklearn.preprocessing import Normalizer, LabelEncoder, MinMaxScaler, StandardScaler
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor

from tensorflow import keras as keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense, Flatten, Dropout, BatchNormalization, Activation
from pandas import read_excel, DataFrame, Series
from keras.wrappers.scikit_learn import KerasClassifier, KerasRegressor
from tensorflow.keras.models import Sequential
from numpy.random import seed
from scipy import stats
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
#Загружаем первый датасет (базальтопластик) и посмотрим на названия столбцов
df = pd.read_excel(r"C:\Users\Avona\Desktop\Моя ВКР\ltog\itog.xlsx")
df.shape
```

Out[2]:
```
(922, 15)
```

Прогнозируем модуль упругости при растяжении, ГПа

In [3]:
```
#разбиваем на тестовую, тренировочную выборки, выделяя предикторы и целевые переменные
normalizer = Normalizer()
res = normalizer.fit_transform(df)
df_norm_n = pd.DataFrame(res, columns = df.columns)
x_train_2, x_test_2, y_train_2, y_test_2 = train_test_split(
    df_norm_n.loc[:, df_norm_n.columns != 'Модуль упругости при растяжении, ГПа'],
    df['Модуль упругости при растяжении, ГПа']],
        test_size = 0.3,
    random_state = 42)
```

In [4]:
```
# Проверка правильности разбивки
df_norm_n.shape[0] - x_train_2.shape[0] - x_test_2.shape[0]
```

Out[4]:
```
0
```

In [5]:
```
x_train_2.head()
```

Out[5]:

	Unnamed: 0.1	Unnamed: 0	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температура вспышки, С_2	Поверхностная плотность, г/м2	Прочность при растяжении, МПа	Потребление смолы, г/м2	Угол нашивки	Шаг нашивки	Плотность нашивки
481	0.156444	0.156444	0.000515	0.574468	0.128874	0.020477	0.006099	0.069680	0.181471	0.748657	0.071227	0.000283	0.002229	0.015177
650	0.215475	0.215475	0.000410	0.587940	0.221451	0.025956	0.006996	0.071984	0.077478	0.704786	0.054136	0.000293	0.001286	0.017439
483	0.159163	0.159163	0.000565	0.570410	0.166143	0.042288	0.006892	0.094073	0.129399	0.752680	0.040423	0.000287	0.002192	0.018444
355	0.108012	0.108012	0.000901	0.539490	0.302537	0.024182	0.006533	0.069661	0.158041	0.748787	0.049559	0.000000	0.001753	0.012393
850	0.280412	0.280412	0.000526	0.546876	0.242346	0.031364	0.006902	0.082237	0.087759	0.680025	0.079257	0.000296	0.002539	0.015771

In [6]:
```
y_train_2
```

Out[6]:

	Модуль упругости при растяжении, ГПа
481	69.573625
650	80.691499
483	71.887367
355	68.314525
850	72.997468
...	...
106	74.519119
270	70.325533
860	77.995289
435	70.199234
102	72.625213

645 rows × 1 columns

In [7]:
```
y_train_2.shape
```

Out[7]:
```
(645, 1)
```

In [8]:
```
# функция для сравнения результатов предсказаний с моделью, выдающей среднее значение по тестовой выборке
def mean_model(y_test_2):
    return [np.mean(y_test_2) for _ in range(len(y_test_2))]
y_2_pred_mean = mean_model(y_test_2)
```

In [9]:
```
#Проверка различных моделей при стандартных параметрах
# Метод опорных векторов - 1
```

In [10]:
```
svr2 = make_pipeline(StandardScaler(), SVR(kernel = 'rbf', C = 500.0, epsilon = 1.0))
#обучаем модель
svr2.fit(x_train_2, np.ravel(y_train_2))
#Вычисляем коэффициент детерминации
y_pred_svr2 = svr2.predict(x_test_2)
mae_svr2 = mean_absolute_error(y_pred_svr2, y_test_2)
mse_svr_elast2 = mean_squared_error(y_test_2,y_pred_svr2)
print('Support Vector Regression Results Train:')
print("Test score: {:.2f}".format(svr2.score(x_train_2, y_train_2))) # Скор для тренировочной выборки
print('Support Vector Regression Results:')
print('SVR_MAE:', round(mean_absolute_error(y_test_2, y_pred_svr2)))
print('SVR_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test_2, y_pred_svr2)))
print('SVR_MSE: {:.2f}'.format(mse_svr_elast2))
print('SVR_RMSE: {:.2f}'.format (np.sqrt(mse_svr_elast2)))
print("Test score: {:.2f}".format(svr2.score(x_test_2, y_test_2))) # Скор для тестовой выборки
```

```
Support Vector Regression Results Train:
Test score: 0.90
Support Vector Regression Results:
SVR_MAE: 3
SVR_MAPE: 0.05
SVR_MSE: 18.69
SVR_RMSE: 4.32
Test score: -0.89
```

In [11]:
```
#Результаты модели, выдающей среднее значение
mse_lin_elast2_mean = mean_squared_error(y_test_2, y_2_pred_mean)
print("MAE for mean target: ", mean_absolute_error(y_test_2, y_2_pred_mean))
print("MSE for mean target: ", mse_lin_elast2_mean)
print("RMSE for mean target: ", np.sqrt(mse_lin_elast2_mean))
```

```
MAE for mean target:  2.578499535756179
MSE for mean target:  9.918160742106828
RMSE for mean target:  3.148072543971442
```

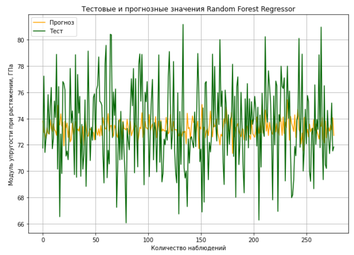
In [12]:
```
plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Support Vector Regression")
plt.plot(y_pred_svr2, label = "Прогноз", color = "orange")
plt.plot(y_test_2.values, label = "Тест", color = "green")
plt.xlabel("Количество наблюдений")
plt.ylabel("Модуль упругости при растяжении, ГПа")
plt.legend()
plt.grid(True);
```

In [13]:
```
# Метод случайного леса - Random Forest Regressor - 2
```

In [14]:
```
#построение модели и бузуализация метода случайный лес
rfr2 = RandomForestRegressor(n_estimators = 15,max_depth = 7, random_state = 33)
rfr2.fit(x_train_2, y_train_2.values)
y2_pred_forest = rfr2.predict(x_test_2)
mae_rfr2 = mean_absolute_error(y2_pred_forest, y_test_2)
mse_rfr_elast2 = mean_squared_error(y_test_2,y2_pred_forest)
print('Random Forest Regressor Results Train:')
print("Test score: {:.2f}".format(rfr2.score(x_train_2, y_train_2))) # Скор для тренировочной выборки
print('Random Forest Regressor Results:')
print('RF_MAE: ', round(mean_absolute_error(y_test_2, y2_pred_forest)))
print('RF_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test_2, y2_pred_forest)))
print('RF_MSE: {:.2f}'.format(mse_rfr_elast2))
print("RF_RMSE: {:.2f}".format (np.sqrt(mse_rfr_elast2)))
print("Test score: {:.2f}".format(rfr2.score(x_test_2, y_test_2))) # Скор для тестовой выборки
```

```
Random Forest Regressor Results Train:
Test score: 0.40
Random Forest Regressor Results:
RF_MAE:  3
RF_MAPE: 0.04
RF_MSE: 10.47
RF_RMSE: 3.24
Test score: -0.06
```

In [15]:
```
plt.figure(figsize=(10, 7))
plt.title("Тестовые и прогнозные значения Random Forest Regressor")
plt.plot(y2_pred_forest, label = "Прогноз", color = "orange")
plt.plot(y_test_2.values, label = "Тест", color = 'darkgreen')
plt.xlabel("Количество наблюдений")
plt.ylabel("Модуль упругости при растяжении, ГПа")
plt.legend()
plt.grid(True);
```

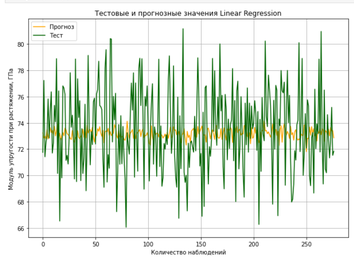


In [16]:
```
#Метод линейной регрессии - Linear Regression - 3
```

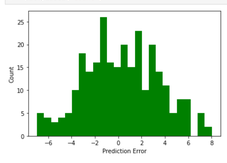
In [17]:
```
#построение модели и бузуализация линейной регрессии
lr2 = LinearRegression()
lr2.fit(x_train_2, y_train_2)
y_pred_lr2 = lr2.predict(x_test_2)
mae_lr2 = mean_absolute_error(y_pred_lr2, y_test_2)
mse_lin_elast2 = mean_squared_error(y_test_2, y_pred_lr2)
print('Linear Regression Results Train:') # Скор для тренировочной выборки
print("Test score: {:.2f}".format(lr2.score(x_train_2, y_train_2)))
print('Linear Regression Results:')
print('lr_MAE: ', round(mean_absolute_error(y_test_2, y_pred_lr2)))
print('lr_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test_2, y_pred_lr2)))
print('lr_MSE: {:.2f}'.format(mse_lin_elast2))
print("lr_RMSE: {:.2f}".format (np.sqrt(mse_lin_elast2)))
print("Test score: {:.2f}".format(lr2.score(x_test_2, y_test_2))) # Скор для тестовой выборки
```

```
Linear Regression Results Train:
Test score: 0.02
Linear Regression Results:
lr_MAE:  3
lr_MAPE: 0.04
lr_MSE: 10.18
lr_RMSE: 3.19
Test score: -0.03
```

In [18]:
```
plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Linear Regression")
plt.plot(y_pred_lr2, label = "Прогноз", color = 'orange')
plt.plot(y_test_2.values, label = "Тест", color = 'darkgreen')
plt.xlabel("Количество наблюдений")
plt.ylabel("Модуль упругости при растяжении, ГПа")
plt.legend()
plt.grid(True);
```



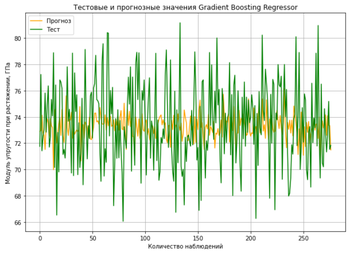
In [19]:
```
error = y_test_2 - y_pred_lr2
plt.hist(error, bins = 25, color = "g")
plt.xlabel('Prediction Error')
_ = plt.ylabel('Count')
```



In [20]:
```
gbr2 = make_pipeline(StandardScaler(), GradientBoostingRegressor())
gbr2.fit(x_train_2, np.ravel(y_train_2))
y_pred_gbr2 = gbr2.predict(x_test_2)
mae_gbr2 = mean_absolute_error(y_pred_gbr2, y_test_2)
mse_gbr_elast2 = mean_squared_error(y_test_2,y_pred_gbr2)
print('Gradient Boosting Regressor Results Train:')
print("Test score: {:.2f}".format(gbr2.score(x_train_2, y_train_2))) # Скор для тренировочной выборки
print('Gradient Boosting Regressor Results:')
print('GBR_MAE: ', round(mean_absolute_error(y_test_2, y_pred_gbr2)))
print('GBR_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test_2, y_pred_gbr2)))
print('GBR_MSE: {:.2f}'.format(mse_gbr_elast2))
print("GBR_RMSE: {:.2f}".format (np.sqrt(mse_gbr_elast2)))
print("Test score: {:.2f}".format(gbr2.score(x_test_2, y_test_2)))# Скор для тестовой выборки
```

```
Gradient Boosting Regressor Results Train:
Test score: 0.53
Gradient Boosting Regressor Results:
GBR_MAE:  3
GBR_MAPE: 0.04
GBR_MSE: 10.81
GBR_RMSE: 3.29
Test score: -0.09
```

In [21]:
```
plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Gradient Boosting Regressor")
plt.plot(y_pred_gbr2, label = "Прогноз", color = "orange")
plt.plot(y_test_2.values, label = "Тест", color = "green")
plt.xlabel("Количество наблюдений")
plt.ylabel("Модуль упругости при растяжении, ГПа")
plt.legend()
plt.grid(True);
```

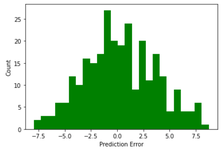


In [22]:
```
# Метод К ближайших соседей - K Neighbors Regressor - 5
knn2 = KNeighborsRegressor(n_neighbors=5)
knn2.fit(x_train_2, y_train_2)
y_pred_knn2 = knn2.predict(x_test_2)
mae_knn2 = mean_absolute_error(y_pred_knn2, y_test_2)
mse_knn_elast2 = mean_squared_error(y_test_2,y_pred_knn2)
print('K Neighbors Regressor  Results Train:')
print("Test score: {:.2f}".format(knn2.score(x_train_2, y_train_2)))# Скор для тренировочной выборки
print('K Neighbors Regressor  Results:')
print('KNN_MAE: ', round(mean_absolute_error(y_test_2, y_pred_knn2)))
print('KNN_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test_2, y_pred_knn2)))
print('KNN_MSE: {:.2f}'.format(mse_knn_elast2))
print('KNN_RMSE: {:.2f}'.format (np.sqrt(mse_knn_elast2)))
print("Test score: {:.2f}".format(knn2.score(x_test_2, y_test_2)))# Скор для тестовой выборки
```

```
K Neighbors Regressor  Results Train:
Test score: 0.24
K Neighbors Regressor  Results:
KNN_MAE:  3
KNN_MAPE: 0.04
KNN_MSE: 11.88
KNN_RMSE: 3.45
Test score: -0.20
```

In [23]:
```
plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения K Neighbors Regressor")
plt.plot(y_pred_knn2, label = "Прогноз", color = 'orange')
plt.plot(y_test_2.values, label = "Тест", color = 'darkgreen')
plt.xlabel("Количество наблюдений")
plt.ylabel("Модуль упругости при растяжении, ГПа")
plt.legend()
plt.grid(True);
```

In [24]:
```
#Визуализация гистограммы распределения ошибки
error = y_test_2 - y_pred_knn2
plt.hist(error, bins = 25, color = "g")
plt.xlabel('Prediction Error')
_ = plt.ylabel('Count')
```



In [25]:
```
#Деревья решений - Decision Tree Regressor - 6

dtr2 = DecisionTreeRegressor()
dtr2.fit(x_train_2, y_train_2.values)
y_pred_dtr2 = dtr2.predict(x_test_2)
mae_dtr2 = mean_absolute_error(y_pred_dtr2, y_test_2)
mse_dtr_elast2 = mean_squared_error(y_test_2,y_pred_dtr2)
print('Decision Tree Regressor Results Train:')
print("Test score: {:.2f}".format(dtr2.score(x_train_2, y_train_2)))# Скор для тренировочной выборки
print('Decision Tree Regressor Results:')
print('DTR_MAE: ', round(mean_absolute_error(y_test_2, y_pred_dtr2)))
print('DTR_MSE: {:.2f}'.format(mse_dtr_elast2))
print('DTR_RMSE: {:.2f}'.format (np.sqrt(mse_dtr_elast2)))
print('DTR_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test_2, y_pred_dtr2)))
print("Test score: {:.2f}".format(dtr2.score(x_test_2, y_test_2)))# Скор для тестовой выборки
```

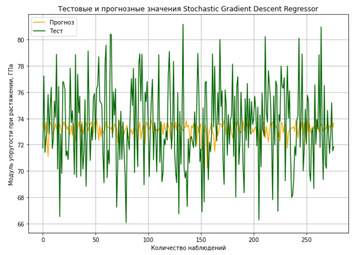
```
Decision Tree Regressor Results Train:
Test score: 1.00
Decision Tree Regressor Results:
DTR_MAE:  4
DTR_MSE: 19.90
DTR_RMSE: 4.46
DTR_MAPE: 0.05
Test score: -1.01
```

In [26]:
```
plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Decision Tree Regressor")
plt.plot(y_pred_dtr2, label = "Прогноз", color = 'orange')
plt.plot(y_test_2.values, label = "Тест", color = 'darkgreen')
plt.xlabel("Количество наблюдений")
plt.ylabel("Модуль упругости при растяжении, ГПа")
plt.legend()
plt.grid(True);
```


In [27]:
```
# Стохастический градиентный спуск (SGD) - Stochastic Gradient Descent Regressor - 7

sdg2 = SGDRegressor()
sdg2.fit(x_train_2, y_train_2)
y_pred_sdg2 = sdg2.predict(x_test_2)
mae_sdg2 = mean_absolute_error(y_pred_sdg2, y_test_2)
mse_sdg_elast2 = mean_squared_error(y_test_2,y_pred_sdg2)
print('Stochastic Gradient Descent Regressor Results Train:')
print("Test score: {:.2f}".format(sdg2.score(x_train_2, y_train_2)))# Скор для тренировочной выборки
print('Stochastic Gradient Descent Regressor Results:')
print('SGD_MAE: ', round(mean_absolute_error(y_test_2, y_pred_sdg2)))
print('SGD_MSE: {:.2f}'.format(mse_sdg_elast2))
print('SGD_RMSE: {:.2f}'.format (np.sqrt(mse_sdg_elast2)))
print('SGD_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test_2, y_pred_sdg2)))
print("Test score: {:.2f}".format(sdg2.score(x_test_2, y_test_2)))# Скор для тестовой выборки
```

```
Stochastic Gradient Descent Regressor Results Train:
Test score: -0.01
Stochastic Gradient Descent Regressor Results:
SGD_MAE:  3
SGD_MSE: 10.27
SGD_RMSE: 3.20
SGD_MAPE: 0.04
Test score: -0.04
```

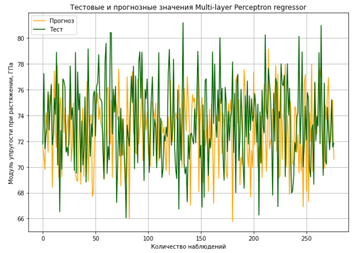
In [28]:
```
plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Stochastic Gradient Descent Regressor")
plt.plot(y_pred_sdg2, label = "Прогноз", color = 'orange')
plt.plot(y_test_2.values, label = "Тест", color = 'darkgreen')
plt.xlabel("Количество наблюдений")
plt.ylabel("Модуль упругости при растяжении, ГПа")
plt.legend()
plt.grid(True);
```



In [29]:
```
# Многослойный перцептрон - Multi-Layer Perceptron regressor - 8

mlp2 = MLPRegressor(random_state = 1, max_iter = 500)
mlp2.fit(x_train_2, y_train_2)
y_pred_mlp2 = mlp2.predict(x_test_2)
mae_mlp2 = mean_absolute_error(y_pred_mlp2, y_test_2)
mse_mlp_elast2 = mean_squared_error(y_test_2,y_pred_mlp2)
print('Multi-layer Perceptron regressor Results Train:')
print("Test score: {:.2f}".format(mlp2.score(x_train_2, y_train_2)))# Скор для тренировочной выборки
print('Multi-layer Perceptron regressor Results:')
print('SGD_MAE: ', round(mean_absolute_error(y_test_2, y_pred_mlp2)))
print('SGD_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test_2, y_pred_mlp2)))
print('SGD_MSE: {:.2f}'.format(mse_mlp_elast2))
print("SGD_RMSE: {:.2f}".format (np.sqrt(mse_mlp_elast2)))
print("Test score: {:.2f}".format(mlp2.score(x_test_2, y_test_2)))# Скор для тестовой выборки
```

```
Multi-layer Perceptron regressor Results Train:
Test score: -0.77
Multi-layer Perceptron regressor Results:
SGD_MAE:  3
SGD_MAPE: 0.05
SGD_MSE: 17.30
SGD_RMSE: 4.16
Test score: -0.75
```

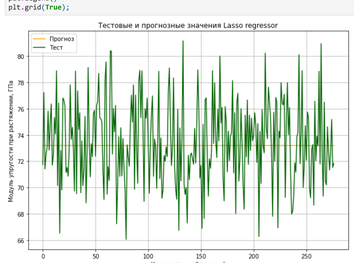
In [30]:
```
plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Multi-layer Perceptron regressor")
plt.plot(y_pred_mlp2, label = "Прогноз", color = 'orange')
plt.plot(y_test_2.values, label = "Тест", color = 'darkgreen')
plt.xlabel("Количество наблюдений")
plt.ylabel("Модуль упругости при растяжении, ГПа")
plt.legend()
plt.grid(True);
```



In [31]:
```
# Лассо регрессия - the Lasso - 9

clf2 = linear_model.Lasso(alpha = 0.1)
clf2.fit(x_train_2, y_train_2)
y_pred_clf2 = clf2.predict(x_test_2)
mae_clf2 = mean_absolute_error(y_pred_clf2, y_test_2)
mse_clf_elast2 = mean_squared_error(y_test_2,y_pred_clf2)
print('Lasso regressor Results Train:')
print("Test score: {:.2f}".format(clf2.score(x_train_2, y_train_2)))# Скор для тренировочной выборки
print('Lasso regressor Results:')
print('SGD_MAE: ', round(mean_absolute_error(y_test_2, y_pred_clf2)))
print('SGD_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test_2, y_pred_clf2)))
print('SGD_MSE: {:.2f}'.format(mse_clf_elast2))
print("SGD_RMSE: {:.2f}".format (np.sqrt(mse_clf_elast2)))
print("Test score: {:.2f}".format(clf2.score(x_test_2, y_test_2)))# Скор для тестовой выборки
```

```
Lasso regressor Results Train:
Test score: 0.00
Lasso regressor Results:
SGD_MAE:  3
SGD_MAPE: 0.04
SGD_MSE: 10.00
SGD_RMSE: 3.16
Test score: -0.01
```

In [32]:
```
plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Lasso regressor")
plt.plot(y_pred_clf2, label = "Прогноз", color = 'orange')
plt.plot(y_test_2.values, label = "Тест", color = 'darkgreen')
plt.xlabel("Количество наблюдений")
plt.ylabel("Модуль упругости при растяжении, ГПа")
plt.legend()
plt.grid(True);
```



In [33]:
```
#сробним наши модели по метрике MAE
mae_df2 = {'Perpeccop': ['Support Vector', 'RandomForest', 'Linear Regression', 'GradientBoosting', 'KNeighbors', 'DecisionTree', 'SGD', 'MLP', 'Lasso'], 'MAE': [mae_svr2, mae_rfr2, mae_lr2, mae_gbr2, mae_knr2, mae_dtr2, mae_sdg2, mae_mlp2, mae_clf2]}
mae_df2 = pd.DataFrame(mae_df2)
```

In [34]:
```
# Проведем поиск  по сетке гиперпараметров с перекрестной проверкой, количество блоков рабно 10 (cv = 10), для
# модели случайного леса - Random Forest Regressor - 2

parametrs = { 'n_estimators': [200, 300],
              'max_depth': [9, 15],
              'max_features': ['auto'],
              'criterion': ['mse'] }
grid21 = GridSearchCV(estimator = rfr2, param_grid  = parametrs, cv=10)
grid21.fit(x_train_2, y_train_2)
```

Out[34]:
```
GridSearchCV(cv=10,
             estimator=RandomForestRegressor(max_depth=7, n_estimators=15,
                                             random_state=33),
             param_grid={'criterion': ['mse'], 'max_depth': [9, 15],
                         'max_features': ['auto'], 'n_estimators': [200, 300]})
```

In [35]:
```
grid21.best_params_
```

Out[35]:
```
{'criterion': 'mse',
 'max_depth': 9,
 'max_features': 'auto',
 'n_estimators': 200}
```

In [36]:
```
#Выводим гиперпараметры для оптимальной модели
print(grid21.best_estimator_)
knr_u = grid21.best_estimator_
print(f'R2-score RFR для модуля упругости при растяжении: {knr_u.score(x_test_2, y_test_2).round(3)}')
```

```
RandomForestRegressor(criterion='mse', max_depth=9, n_estimators=200,
                      random_state=33)
R2-score RFR для модуля упругости при растяжении: -0.035
```

In [37]:
```
#подстабим оптимальные гиперпараметры в нашу модель случайного леса
rfr21_grid = RandomForestRegressor(n_estimators=200, criterion='mse', max_depth=15, max_features='auto')
#Обучаем модель
rfr21_grid.fit(x_train_2, y_train_2)

predictions_rfr21_grid = rfr21_grid.predict(x_test_2)
#Оценибаем точность на тестовом наборе
mae_rfr21_grid = mean_absolute_error(predictions_rfr21_grid, y_test_2)
mae_rfr21_grid
```

Out[37]:
```
2.6288324927708726
```

In [38]:
```
new_row_in_mae_df = {'Перрессор': 'RandomForest1_GridSearchCV', 'MAE': mae_rfr21_grid}

mae_df = mae_df2.append(new_row_in_mae_df, ignore_index = True)
```

In [39]:
```
mae_df
```

Out[39]:

	Перрессор	MAE
0	Support Vector	3.467880
1	RandomForest	2.621567
2	Linear Regression	2.612273
3	GradientBoosting	2.649635
4	KNeighbors	2.789287
5	DecisionTree	3.628268
6	SGD	2.613983
7	MLP	3.338349
8	Lasso	2.580193
9	RandomForest1_GridSearchCV	2.628832

In [40]:
```
# Проведем поиск  по сетке гиперпараметров с перекрестной проверкой, количество блоков рабно 10 (cv = 10), для
# Метода К ближайших соседей - K Neighbors Regressor - 5
knn21 = KNeighborsRegressor()
knn21_params = {'n_neighbors' : range(1, 301, 2),
          'weights' : ['uniform', 'distance'],
          'algorithm' : ['auto', 'ball_tree', 'kd_tree', 'brute']
         }
#Запустим обучение модели. В качестве оценки модели будем использовать коэффициент детерминации (R^2)
# Если R2<0, это значит, что разработанная модель даёт прогноз даже хуже, чем простое усреднение.
gs21 = GridSearchCV(knn21, knn21_params, cv = 10, verbose = 1, n_jobs=-1, scoring = 'r2')
gs21.fit(x_train_2, y_train_2)
knn_21 = gs21.best_estimator_
gs21.best_params_
```

```
Fitting 10 folds for each of 1200 candidates, totalling 12000 fits
```

Out[40]:
```
{'algorithm': 'auto', 'n_neighbors': 269, 'weights': 'uniform'}
```

In [41]:
```
#Выводим гиперпараметры для оптимальной модели
print(gs21.best_estimator_)
gs121 = gs21.best_estimator_
print(f'R2-score KNR для модуля упругости при растяжении: {gs121.score(x_test_2, y_test_2).round(3)}')
```

```
KNeighborsRegressor(n_neighbors=269)
R2-score KNR для модуля упругости при растяжении: -0.013
```

In [42]:
```
#подстабим оптимальные гиперпараметры в нашу модель метода к ближайших соседей
knn21_grid = KNeighborsRegressor(algorithm = 'brute', n_neighbors = 7, weights = 'distance')
#Обучаем модель
knn21_grid.fit(x_train_2, y_train_2)

predictions_knn21_grid = knn21_grid.predict(x_test_2)
#Оценибаем точность на тестовом наборе
mae_knn21_grid = mean_absolute_error(predictions_knn21_grid, y_test_2)
mae_knn21_grid
```

Out[42]:
```
2.745343552900663
```

In [43]:
```
new_row_in_mae_df = {'Перрессор': 'KNeighbors1_GridSearchCV', 'MAE': mae_knn21_grid}

mae_df = mae_df.append(new_row_in_mae_df, ignore_index=True)
mae_df
```

Out[43]:

	Перрессор	MAE
0	Support Vector	3.467880
1	RandomForest	2.621567
2	Linear Regression	2.612273
3	GradientBoosting	2.649635
4	KNeighbors	2.789287
5	DecisionTree	3.628268
6	SGD	2.613983
7	MLP	3.338349
8	Lasso	2.580193
9	RandomForest1_GridSearchCV	2.628832
10	KNeighbors1_GridSearchCV	2.745344

In [44]:
```
# Проведем поиск  по сетке гиперпараметров с перекрестной проверкой, количество блоков рабно 10 (cv = 10), для
#Деревья решений - Decision Tree Regressor - 6
criterion21 = ['squared_error', 'friedman_mse', 'absolute_error', 'poisson']
splitter21 = ['best', 'random']
max_depth21 = [3,5,7,9,11]
min_samples_leaf21 = [100,150,200]
min_samples_split21 = [200,250,300]
max_features21 = ['auto', 'sqrt', 'log2']
param_grid21 = {'criterion': criterion21,
                'splitter': splitter21,
                'max_depth': max_depth21,
                'min_samples_split': min_samples_split21,
                'min_samples_leaf': min_samples_leaf21,
                'max_features': max_features21}
#Запустим обучение модели. В качестве оценки модели будем использовать коэффициент детерминации (R^2)
# Если R2<0, это значит, что разработанная модель даёт прогноз даже хуже, чем простое усреднение.
gs21 = GridSearchCV(dtr2, param_grid21, cv = 10, verbose = 1, n_jobs=-1, scoring = 'r2')
gs21.fit(x_train_2, y_train_2)
dtr_21 = gs21.best_estimator_
gs21.best_params_
```

```
Fitting 10 folds for each of 1080 candidates, totalling 10800 fits
```

Out[44]:
```
{'criterion': 'friedman_mse',
 'max_depth': 7,
 'max_features': 'log2',
 'min_samples_leaf': 200,
 'min_samples_split': 250,
 'splitter': 'best'}
```

In [45]:
```
#Выводим гиперпараметры для оптимальной модели
print(gs21.best_estimator_)
gs21 = gs21.best_estimator_
print(f'R2-score DTR для модуля упругости при растяжении: {gs21.score(x_test_2, y_test_2).round(3)}')
```

```
DecisionTreeRegressor(criterion='friedman_mse', max_depth=7,
                      max_features='log2', min_samples_leaf=200,
                      min_samples_split=250)
R2-score DTR для модуля упругости при растяжении: -0.019
```

In [46]:
```
#подстабим оптимальные гиперпараметры в нашу модель метода деревья решений
dtr21_grid = DecisionTreeRegressor(criterion='poisson', max_depth=7, max_features='auto',
                      min_samples_leaf=100, min_samples_split=250)
#Обучаем модель
dtr21_grid.fit(x_train_2, y_train_2)

predictions_dtr21_grid = dtr21_grid.predict(x_test_2)
#Оценибаем точность на тестовом наборе
mae_dtr21_grid = mean_absolute_error(predictions_dtr21_grid, y_test_2)
mae_dtr21_grid
```

Out[46]:
```
2.606181669247976
```

In [47]:
```
new_row_in_mae_df = {'Perpeccop': 'DecisionTree1_GridSearchCV', 'MAE': mae_dtr21_grid}

mae_df = mae_df.append(new_row_in_mae_df, ignore_index=True)
mae_df
```

Out[47]:

	Перрессор	MAE
0	Support Vector	3.467880
1	RandomForest	2.621567
2	Linear Regression	2.612273
3	GradientBoosting	2.649635
4	KNeighbors	2.789287
5	DecisionTree	3.628268
6	SGD	2.613983
7	MLP	3.338349
8	Lasso	2.580193
9	RandomForest1_GridSearchCV	2.628832
10	KNeighbors1_GridSearchCV	2.745344
11	DecisionTree1_GridSearchCV	2.606182

In [49]:
```
pipe2 = Pipeline([('preprocessing', StandardScaler()), ('regressor', SVR())])
param_grid2 = [
{'regressor': [SVR()], 'preprocessing': [StandardScaler(), MinMaxScaler(), None],
'regressor__gamma': [0.001, 0.01, 0.1, 1, 10, 100],
'regressor__C': [0.001, 0.01, 0.1, 1, 10, 100]},
{'regressor': [RandomForestRegressor(n_estimators=100)],
'preprocessing':  [StandardScaler(), MinMaxScaler(), None]},
{'regressor': [LinearRegression()], 'preprocessing':  [StandardScaler(), MinMaxScaler(), None]},
{'regressor': [GradientBoostingRegressor()], 'preprocessing':  [StandardScaler(), MinMaxScaler(), None]},
{'regressor': [KNeighborsRegressor()], 'preprocessing':  [StandardScaler(), MinMaxScaler(), None]},
{'regressor': [DecisionTreeRegressor()], 'preprocessing':  [StandardScaler(), MinMaxScaler(), None]},
{'regressor': [SGDRegressor()], 'preprocessing':  [StandardScaler(), MinMaxScaler(), None]},
{'regressor': [MLPRegressor(random_state=1, max_iter=500)], 'preprocessing':  [StandardScaler(), MinMaxScaler(), None]},
{'regressor': [linear_model.Lasso(alpha=0.1)], 'preprocessing':  [StandardScaler(), MinMaxScaler(), None]},]
grid2 = GridSearchCV(pipe2, param_grid2, cv=10)
grid2.fit(x_train_2, np.ravel(y_train_2))
print("Наилучшие параметры:\n{}\n".format(grid2.best_params_))
print("Наилучшее значение правильности перекрестной проверки: {:.2f}".format(grid2.best_score_))
print("Правильность на тестовом наборе: {:.2f}".format(grid2.score(x_test_2, y_test_2)))
```

```
Наилучшие параметры:
{'preprocessing': MinMaxScaler(), 'regressor': SVR(C=10, gamma=100), 'regressor__C': 10, 'regressor__gamma': 100}

Наилучшее значение правильности перекрестной проверки: -0.01
Правильность на тестовом наборе: -0.01
```

In [50]:
```
print("Наилучшая модель:\n{}".format(grid2.best_estimator_))
```

```
Наилучшая модель:
Pipeline(steps=[('preprocessing', MinMaxScaler()),
                ('regressor', SVR(C=10, gamma=100))])
```

После обучения моделей была проведена оценка точности этих моделей на обучающей и тестовых выборках. В качестве параметра оценки модели использовалась средняя абсолютная ошибка (MAE). Обе модели даже на тренировочном датасете не смогли обучиться и приблизиться к исходным данным. Поэтому ошибка на тестовом датасете выше.

Написать нейронную сеть, которая будет рекомендовать соотношение матрица-наполнитель.

In [51]:
```
# Сформируем входы и выход для модели

tv = df['Соотношение матрица-наполнитель']
tr_v = df.loc[:, df.columns != 'Соотношение матрица-наполнитель']

# Разбиваем выборки на обучающие и тестовую
x_train, x_test, y_train, y_test = train_test_split(tr_v, tv, test_size = 0.3, random_state = 14)
```

In [52]:
```
# Нормализуем данные

x_train_n = tf.keras.layers.Normalization(axis =-1)
x_train_n.adapt(np.array(x_train))
```

In [53]:
```
def create_model(lyrs=[32], act='softmax', opt='SGD', dr=0.1):

    seed = 7
    np.random.seed(seed)
    tf.random.set_seed(seed)

    model = Sequential()
    model.add(Dense(lyrs[0], input_dim=x_train.shape[1], activation=act))
    for i in range(1,len(lyrs)):
        model.add(Dense(lyrs[i], activation=act))

    model.add(Dropout(dr))
    model.add(Dense(3, activation='tanh'))  # выходной слой

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

    return model
```

In [54]:
```
# строим модель
model = KerasClassifier(build_fn=create_model, verbose=0)

# определяем параметры
batch_size = [4, 10, 20, 50, 100]
epochs = [10, 50, 100, 200, 300]
param_grid = dict(batch_size=batch_size, epochs=epochs)

# поиск оптимальных параметров
grid = GridSearchCV(estimator=model,
                    param_grid=param_grid,
                    cv=10,
                    verbose=1, n_jobs=-1)

grid_result = grid.fit(x_train, y_train)
```

```
Fitting 10 folds for each of 25 candidates, totalling 250 fits
```

In [55]:
```
# результаты
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.001538 using {'batch_size': 4, 'epochs': 10}
0.001538 (0.004615) with: {'batch_size': 4, 'epochs': 10}
0.001538 (0.004615) with: {'batch_size': 4, 'epochs': 50}
0.001538 (0.004615) with: {'batch_size': 4, 'epochs': 100}
0.001538 (0.004615) with: {'batch_size': 4, 'epochs': 200}
0.001538 (0.004615) with: {'batch_size': 4, 'epochs': 300}
0.001538 (0.004615) with: {'batch_size': 10, 'epochs': 10}
0.001538 (0.004615) with: {'batch_size': 10, 'epochs': 50}
0.001538 (0.004615) with: {'batch_size': 10, 'epochs': 100}
0.001538 (0.004615) with: {'batch_size': 10, 'epochs': 200}
0.001538 (0.004615) with: {'batch_size': 10, 'epochs': 300}
0.001538 (0.004615) with: {'batch_size': 20, 'epochs': 10}
0.001538 (0.004615) with: {'batch_size': 20, 'epochs': 50}
0.001538 (0.004615) with: {'batch_size': 20, 'epochs': 100}
0.001538 (0.004615) with: {'batch_size': 20, 'epochs': 200}
0.001538 (0.004615) with: {'batch_size': 20, 'epochs': 300}
0.001538 (0.004615) with: {'batch_size': 50, 'epochs': 10}
0.001538 (0.004615) with: {'batch_size': 50, 'epochs': 50}
0.001538 (0.004615) with: {'batch_size': 50, 'epochs': 100}
0.001538 (0.004615) with: {'batch_size': 50, 'epochs': 200}
0.001538 (0.004615) with: {'batch_size': 50, 'epochs': 300}
0.001538 (0.004615) with: {'batch_size': 100, 'epochs': 10}
0.001538 (0.004615) with: {'batch_size': 100, 'epochs': 50}
0.001538 (0.004615) with: {'batch_size': 100, 'epochs': 100}
0.001538 (0.004615) with: {'batch_size': 100, 'epochs': 200}
0.001538 (0.004615) with: {'batch_size': 100, 'epochs': 300}
```

In [56]:
```
model = KerasClassifier(build_fn=create_model, epochs=50, batch_size=4, verbose=0)

optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Nadam']
param_grid = dict(opt=optimizer)

grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=10, verbose=2)
grid_result = grid.fit(x_train, y_train)
```

```
Fitting 10 folds for each of 6 candidates, totalling 60 fits
[CV] END ..................................................opt=SGD; total time=  17.3s
[CV] END ..................................................opt=SGD; total time=  12.7s
[CV] END ..................................................opt=SGD; total time=  13.7s
[CV] END ..................................................opt=SGD; total time=  10.8s
[CV] END ..................................................opt=SGD; total time=  10.6s
[CV] END ..................................................opt=SGD; total time=  11.9s
[CV] END ..................................................opt=SGD; total time=  11.9s
[CV] END ..................................................opt=SGD; total time=  12.0s
[CV] END ..................................................opt=SGD; total time=  13.2s
[CV] END ..................................................opt=SGD; total time=  12.2s
[CV] END ..................................................opt=RMSprop; total time=  10.9s
[CV] END ..................................................opt=RMSprop; total time=  10.9s
[CV] END ..................................................opt=RMSprop; total time=  10.4s
[CV] END ..................................................opt=RMSprop; total time=  11.1s
[CV] END ..................................................opt=RMSprop; total time=  11.2s
[CV] END ..................................................opt=RMSprop; total time=  23.7s
[CV] END ..................................................opt=RMSprop; total time=  24.2s
[CV] END ..................................................opt=RMSprop; total time=  23.9s
[CV] END ..................................................opt=RMSprop; total time=  24.8s
[CV] END ..................................................opt=RMSprop; total time=  26.1s
[CV] END ..................................................opt=Adagrad; total time=  16.2s
[CV] END ..................................................opt=Adagrad; total time=  10.8s
[CV] END ..................................................opt=Adagrad; total time=  11.1s
[CV] END ..................................................opt=Adagrad; total time=  10.1s
[CV] END ..................................................opt=Adagrad; total time=  10.3s
[CV] END ..................................................opt=Adagrad; total time=  11.6s
[CV] END ..................................................opt=Adagrad; total time=  11.8s
[CV] END ..................................................opt=Adagrad; total time=  11.3s
[CV] END ..................................................opt=Adagrad; total time=  12.0s
[CV] END ..................................................opt=Adagrad; total time=  14.6s
[CV] END ..................................................opt=Adadelta; total time=  13.1s
[CV] END ..................................................opt=Adadelta; total time=  13.5s
[CV] END ..................................................opt=Adadelta; total time=  13.4s
[CV] END ..................................................opt=Adadelta; total time=  11.1s
[CV] END ..................................................opt=Adadelta; total time=  10.1s
[CV] END ..................................................opt=Adadelta; total time=  12.0s
[CV] END ..................................................opt=Adadelta; total time=  14.1s
[CV] END ..................................................opt=Adadelta; total time=  15.4s
[CV] END ..................................................opt=Adadelta; total time=  13.6s
[CV] END ..................................................opt=Adadelta; total time=  15.8s
[CV] END ..................................................opt=Adam; total time=  12.9s
[CV] END ..................................................opt=Adam; total time=  13.6s
[CV] END ..................................................opt=Adam; total time=  12.6s
[CV] END ..................................................opt=Adam; total time=  11.2s
[CV] END ..................................................opt=Adam; total time=  12.4s
[CV] END ..................................................opt=Adam; total time=  12.6s
[CV] END ..................................................opt=Adam; total time=  14.6s
[CV] END ..................................................opt=Adam; total time=  17.1s
[CV] END ..................................................opt=Adam; total time=  13.4s
[CV] END ..................................................opt=Adam; total time=  14.0s
[CV] END ..................................................opt=Nadam; total time=  12.6s
[CV] END ..................................................opt=Nadam; total time=  11.2s
[CV] END ..................................................opt=Nadam; total time=  11.9s
[CV] END ..................................................opt=Nadam; total time=  14.4s
[CV] END ..................................................opt=Nadam; total time=  16.4s
[CV] END ..................................................opt=Nadam; total time=  20.5s
[CV] END ..................................................opt=Nadam; total time=  15.3s
[CV] END ..................................................opt=Nadam; total time=  17.3s
[CV] END ..................................................opt=Nadam; total time=  15.8s
[CV] END ..................................................opt=Nadam; total time=  12.5s
```

In [57]:
```
# результаты
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.001538 using {'opt': 'SGD'}
0.001538 (0.004615) with: {'opt': 'SGD'}
0.001538 (0.004615) with: {'opt': 'RMSprop'}
0.001538 (0.004615) with: {'opt': 'Adagrad'}
0.001538 (0.004615) with: {'opt': 'Adadelta'}
0.001538 (0.004615) with: {'opt': 'Adam'}
0.000000 (0.000000) with: {'opt': 'Nadam'}
```

In [58]:
```
model = KerasClassifier(build_fn=create_model, epochs=50, batch_size=4, verbose=0)

layers = [[8],[16, 4],[32, 8, 3],[12, 6, 3], [64, 64, 3], [128, 64, 16, 3]]
param_grid = dict(lyrs=layers)

grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=10, verbose=2)
grid_result = grid.fit(x_train, y_train)
```

```
Fitting 10 folds for each of 6 candidates, totalling 60 fits
[CV] END ..................................................lyrs=[8]; total time=  11.4s
[CV] END ..................................................lyrs=[8]; total time=  11.4s
[CV] END ..................................................lyrs=[8]; total time=  10.5s
[CV] END ..................................................lyrs=[8]; total time=  11.7s
[CV] END ..................................................lyrs=[8]; total time=  11.2s
[CV] END ..................................................lyrs=[8]; total time=  12.3s
[CV] END ..................................................lyrs=[8]; total time=  11.9s
[CV] END ..................................................lyrs=[8]; total time=  14.0s
[CV] END ..................................................lyrs=[8]; total time=  12.0s
[CV] END ..................................................lyrs=[8]; total time=  11.2s
[CV] END ..................................................lyrs=[16, 4]; total time=  11.5s
[CV] END ..................................................lyrs=[16, 4]; total time=  11.8s
[CV] END ..................................................lyrs=[16, 4]; total time=  11.2s
[CV] END ..................................................lyrs=[16, 4]; total time=  10.5s
[CV] END ..................................................lyrs=[16, 4]; total time=  12.0s
[CV] END ..................................................lyrs=[16, 4]; total time=  13.6s
[CV] END ..................................................lyrs=[16, 4]; total time=  16.3s
[CV] END ..................................................lyrs=[16, 4]; total time=  16.0s
[CV] END ..................................................lyrs=[16, 4]; total time=  13.3s
[CV] END ..................................................lyrs=[16, 4]; total time=  12.8s
[CV] END ..................................................lyrs=[32, 8, 3]; total time=  12.0s
[CV] END ..................................................lyrs=[32, 8, 3]; total time=  11.8s
[CV] END ..................................................lyrs=[32, 8, 3]; total time=  11.1s
[CV] END ..................................................lyrs=[32, 8, 3]; total time=  12.5s
[CV] END ..................................................lyrs=[32, 8, 3]; total time=  15.3s
[CV] END ..................................................lyrs=[32, 8, 3]; total time=  14.4s
[CV] END ..................................................lyrs=[32, 8, 3]; total time=  14.7s
[CV] END ..................................................lyrs=[32, 8, 3]; total time=  12.6s
[CV] END ..................................................lyrs=[32, 8, 3]; total time=  14.1s
[CV] END ..................................................lyrs=[32, 8, 3]; total time=  15.2s
[CV] END ..................................................lyrs=[12, 6, 3]; total time=  11.1s
[CV] END ..................................................lyrs=[12, 6, 3]; total time=  10.8s
[CV] END ..................................................lyrs=[12, 6, 3]; total time=  13.5s
[CV] END ..................................................lyrs=[12, 6, 3]; total time=  11.5s
[CV] END ..................................................lyrs=[12, 6, 3]; total time=  12.4s
[CV] END ..................................................lyrs=[12, 6, 3]; total time=  15.1s
[CV] END ..................................................lyrs=[12, 6, 3]; total time=  13.9s
[CV] END ..................................................lyrs=[12, 6, 3]; total time=  12.3s
[CV] END ..................................................lyrs=[12, 6, 3]; total time=  11.8s
[CV] END ..................................................lyrs=[12, 6, 3]; total time=  12.3s
[CV] END ..................................................lyrs=[64, 64, 3]; total time=  11.5s
[CV] END ..................................................lyrs=[64, 64, 3]; total time=  12.2s
[CV] END ..................................................lyrs=[64, 64, 3]; total time=  13.6s
[CV] END ..................................................lyrs=[64, 64, 3]; total time=  14.0s
[CV] END ..................................................lyrs=[64, 64, 3]; total time=  12.8s
[CV] END ..................................................lyrs=[64, 64, 3]; total time=  14.7s
[CV] END ..................................................lyrs=[64, 64, 3]; total time=  13.3s
[CV] END ..................................................lyrs=[64, 64, 3]; total time=  14.9s
[CV] END ..................................................lyrs=[64, 64, 3]; total time=  14.7s
[CV] END ..................................................lyrs=[64, 64, 3]; total time=  14.1s
[CV] END ..................................................lyrs=[128, 64, 16, 3]; total time=  12.2s
[CV] END ..................................................lyrs=[128, 64, 16, 3]; total time=  12.1s
[CV] END ..................................................lyrs=[128, 64, 16, 3]; total time=  14.1s
[CV] END ..................................................lyrs=[128, 64, 16, 3]; total time=  15.4s
[CV] END ..................................................lyrs=[128, 64, 16, 3]; total time=  12.7s
[CV] END ..................................................lyrs=[128, 64, 16, 3]; total time=  14.0s
[CV] END ..................................................lyrs=[128, 64, 16, 3]; total time=  13.6s
[CV] END ..................................................lyrs=[128, 64, 16, 3]; total time=  13.7s
[CV] END ..................................................lyrs=[128, 64, 16, 3]; total time=  14.7s
[CV] END ..................................................lyrs=[128, 64, 16, 3]; total time=  14.7s
```

In [59]:
```
# результаты
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.004639 using {'lyrs': [128, 64, 16, 3]}
0.000000 (0.000000) with: {'lyrs': [8]}
0.001538 (0.004615) with: {'lyrs': [16, 4]}
0.001538 (0.004615) with: {'lyrs': [32, 8, 3]}
0.001538 (0.004615) with: {'lyrs': [12, 6, 3]}
0.001538 (0.004615) with: {'lyrs': [64, 64, 3]}
0.004639 (0.009877) with: {'lyrs': [128, 64, 16, 3]}
```

In [60]:
```
model = KerasClassifier(build_fn=create_model, epochs=50, batch_size=4, verbose=0)

activation = ['softmax', 'softplus', 'softsign', 'relu', 'tanh', 'sigmoid', 'hard_sigmoid', 'linear']
param_grid = dict(act=activation)

grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=10)
grid_result = grid.fit(x_train, y_train)
```

In [61]:
```
# результаты
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.001538 using {'act': 'softmax'}
0.001538 (0.004615) with: {'act': 'softmax'}
0.001538 (0.004615) with: {'act': 'softplus'}
0.001538 (0.004615) with: {'act': 'softsign'}
0.001538 (0.004615) with: {'act': 'relu'}
0.001538 (0.004615) with: {'act': 'tanh'}
0.001538 (0.004615) with: {'act': 'sigmoid'}
0.001538 (0.004615) with: {'act': 'hard_sigmoid'}
0.001538 (0.004615) with: {'act': 'linear'}
```

In [62]:
```
model = KerasClassifier(build_fn=create_model, epochs=50, batch_size=4, verbose=0)
```

```
drops = [0.0, 0.01, 0.05, 0.1, 0.2, 0.3, 0.5]
param_grid = dict(dr=drops)

grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=10, verbose=2)
grid_result = grid.fit(x_train, y_train)
```

```
Fitting 10 folds for each of 7 candidates, totalling 70 fits
[CV] END ....................................................dr=0.0; total time=  10.3s
[CV] END ....................................................dr=0.0; total time=  11.7s
[CV] END ....................................................dr=0.0; total time=  11.2s
[CV] END ....................................................dr=0.0; total time=  10.9s
[CV] END ....................................................dr=0.0; total time=  11.8s
[CV] END ....................................................dr=0.0; total time=  14.8s
[CV] END ....................................................dr=0.0; total time=  11.9s
[CV] END ....................................................dr=0.0; total time=  13.4s
[CV] END ....................................................dr=0.0; total time=  12.7s
[CV] END ....................................................dr=0.0; total time=  12.0s
[CV] END ...................................................dr=0.01; total time=  10.6s
[CV] END ...................................................dr=0.01; total time=  11.3s
[CV] END ...................................................dr=0.01; total time=  10.2s
[CV] END ...................................................dr=0.01; total time=  11.0s
[CV] END ...................................................dr=0.01; total time=  12.5s
[CV] END ...................................................dr=0.01; total time=  19.9s
[CV] END ...................................................dr=0.01; total time=  12.9s
[CV] END ...................................................dr=0.01; total time=  12.9s
[CV] END ...................................................dr=0.01; total time=  11.7s
[CV] END ...................................................dr=0.01; total time=  13.6s
[CV] END ...................................................dr=0.05; total time=  15.9s
[CV] END ...................................................dr=0.05; total time=  12.2s
[CV] END ...................................................dr=0.05; total time=  10.6s
[CV] END ...................................................dr=0.05; total time=  11.4s
[CV] END ...................................................dr=0.05; total time=  12.1s
[CV] END ...................................................dr=0.05; total time=  14.8s
[CV] END ...................................................dr=0.05; total time=  15.2s
[CV] END ...................................................dr=0.05; total time=  15.4s
[CV] END ...................................................dr=0.05; total time=  13.1s
[CV] END ...................................................dr=0.05; total time=  13.5s
[CV] END ....................................................dr=0.1; total time=  10.8s
[CV] END ....................................................dr=0.1; total time=  11.6s
[CV] END ....................................................dr=0.1; total time=  11.8s
[CV] END ....................................................dr=0.1; total time=  16.1s
[CV] END ....................................................dr=0.1; total time=  14.9s
[CV] END ....................................................dr=0.1; total time=  13.2s
[CV] END ....................................................dr=0.1; total time=  12.0s
[CV] END ....................................................dr=0.1; total time=  12.5s
[CV] END ....................................................dr=0.1; total time=  11.7s
[CV] END ....................................................dr=0.1; total time=  11.9s
[CV] END ....................................................dr=0.2; total time=  10.5s
[CV] END ....................................................dr=0.2; total time=  11.6s
[CV] END ....................................................dr=0.2; total time=  13.3s
[CV] END ....................................................dr=0.2; total time=  12.1s
[CV] END ....................................................dr=0.2; total time=  13.7s
[CV] END ....................................................dr=0.2; total time=  17.3s
[CV] END ....................................................dr=0.2; total time=  14.2s
[CV] END ....................................................dr=0.2; total time=  12.2s
[CV] END ....................................................dr=0.2; total time=  12.9s
[CV] END ....................................................dr=0.2; total time=  13.2s
[CV] END ....................................................dr=0.3; total time=  14.5s
[CV] END ....................................................dr=0.3; total time=  11.9s
[CV] END ....................................................dr=0.3; total time=  12.8s
[CV] END ....................................................dr=0.3; total time=  10.2s
[CV] END ....................................................dr=0.3; total time=  10.3s
[CV] END ....................................................dr=0.3; total time=  11.7s
[CV] END ....................................................dr=0.3; total time=  11.2s
[CV] END ....................................................dr=0.3; total time=  11.4s
[CV] END ....................................................dr=0.3; total time=  12.3s
[CV] END ....................................................dr=0.3; total time=  11.7s
[CV] END ....................................................dr=0.5; total time=   9.8s
[CV] END ....................................................dr=0.5; total time=  10.0s
[CV] END ....................................................dr=0.5; total time=   9.9s
[CV] END ....................................................dr=0.5; total time=  10.1s
[CV] END ....................................................dr=0.5; total time=  10.2s
[CV] END ....................................................dr=0.5; total time=  11.5s
[CV] END ....................................................dr=0.5; total time=  11.4s
[CV] END ....................................................dr=0.5; total time=  12.1s
[CV] END ....................................................dr=0.5; total time=  11.4s
[CV] END ....................................................dr=0.5; total time=  11.2s
```

In [63]:
```
# результаты
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.001538 using {'dr': 0.0}
0.001538 (0.004615) with: {'dr': 0.0}
0.001538 (0.004615) with: {'dr': 0.01}
0.001538 (0.004615) with: {'dr': 0.05}
0.001538 (0.004615) with: {'dr': 0.1}
0.001538 (0.004615) with: {'dr': 0.2}
0.001538 (0.004615) with: {'dr': 0.3}
0.001538 (0.004615) with: {'dr': 0.5}
```

In [64]:
```
# построение окончательной модели
model = create_model(lyrs=[128, 64, 16, 3], dr=0.05)

print(model.summary())
```

```
Model: "sequential_195"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_493 (Dense)           (None, 128)               1920

 dense_494 (Dense)           (None, 64)                8256

 dense_495 (Dense)           (None, 16)                1040

 dense_496 (Dense)           (None, 3)                 51

 dropout_195 (Dropout)       (None, 3)                 0

 dense_497 (Dense)           (None, 3)                 12

=================================================================
Total params: 11,279
Trainable params: 11,279
Non-trainable params: 0
_________________________________________________________________
None
```

In [65]:
```
# обучаем нейросеть, 80/20 CV
model_hist = model.fit(x_train,
    y_train,
    epochs = 100,
    verbose = 1,
    validation_split = 0.2)
```

```
Epoch 1/100
17/17 [==============================] - 1s 18ms/step - loss: 45.4418 - mae: 3.3252 - accuracy: 0.0000e+00 - val_loss: 42.8621 - val_mae: 3.1117 - val_accuracy: 0.0000e+00
Epoch 2/100
17/17 [==============================] - 0s 5ms/step - loss: 40.2136 - mae: 3.2242 - accuracy: 0.0000e+00 - val_loss: 28.3176 - val_mae: 2.8310 - val_accuracy: 0.0000e+00
Epoch 3/100
17/17 [==============================] - 0s 4ms/step - loss: 30.0003 - mae: 3.0047 - accuracy: 0.0000e+00 - val_loss: 27.6395 - val_mae: 2.7770 - val_accuracy: 0.0000e+00
Epoch 4/100
17/17 [==============================] - 0s 4ms/step - loss: 29.3570 - mae: 2.9674 - accuracy: 0.0000e+00 - val_loss: 27.1260 - val_mae: 2.7562 - val_accuracy: 0.0000e+00
Epoch 5/100
17/17 [==============================] - 0s 4ms/step - loss: 28.7412 - mae: 2.9550 - accuracy: 0.0000e+00 - val_loss: 26.6705 - val_mae: 2.7449 - val_accuracy: 0.0000e+00
Epoch 6/100
17/17 [==============================] - 0s 4ms/step - loss: 28.2981 - mae: 2.9462 - accuracy: 0.0000e+00 - val_loss: 26.2300 - val_mae: 2.7450 - val_accuracy: 0.0000e+00
Epoch 7/100
17/17 [==============================] - 0s 4ms/step - loss: 27.8005 - mae: 2.9491 - accuracy: 0.0000e+00 - val_loss: 25.8002 - val_mae: 2.7484 - val_accuracy: 0.0000e+00
Epoch 8/100
17/17 [==============================] - 0s 6ms/step - loss: 27.3542 - mae: 2.9540 - accuracy: 0.0000e+00 - val_loss: 25.3727 - val_mae: 2.7547 - val_accuracy: 0.0000e+00
Epoch 9/100
17/17 [==============================] - 0s 4ms/step - loss: 18.9556 - mae: 2.7619 - accuracy: 0.0000e+00 - val_loss: 9.7200 - val_mae: 2.3389 - val_accuracy: 0.0000e+00
Epoch 10/100
17/17 [==============================] - 0s 3ms/step - loss: 9.7895 - mae: 2.5313 - accuracy: 0.0000e+00 - val_loss: 8.8560 - val_mae: 2.3176 - val_accuracy: 0.0000e+00
Epoch 11/100
17/17 [==============================] - 0s 4ms/step - loss: 8.8622 - mae: 2.5149 - accuracy: 0.0000e+00 - val_loss: 8.0307 - val_mae: 2.3078 - val_accuracy: 0.0000e+00
Epoch 12/100
17/17 [==============================] - 0s 3ms/step - loss: 7.9078 - mae: 2.5101 - accuracy: 0.0000e+00 - val_loss: 7.1886 - val_mae: 2.3041 - val_accuracy: 0.0000e+00
Epoch 13/100
17/17 [==============================] - 0s 4ms/step - loss: 6.9894 - mae: 2.5087 - accuracy: 0.0000e+00 - val_loss: 6.3418 - val_mae: 2.3037 - val_accuracy: 0.0000e+00
Epoch 14/100
17/17 [==============================] - 0s 3ms/step - loss: 6.0750 - mae: 2.5059 - accuracy: 0.0000e+00 - val_loss: 5.4653 - val_mae: 2.3051 - val_accuracy: 0.0000e+00
Epoch 15/100
17/17 [==============================] - 0s 5ms/step - loss: 5.1011 - mae: 2.5087 - accuracy: 0.0000e+00 - val_loss: 4.5913 - val_mae: 2.3075 - val_accuracy: 0.0000e+00
Epoch 16/100
17/17 [==============================] - 0s 5ms/step - loss: 4.0804 - mae: 2.5125 - accuracy: 0.0000e+00 - val_loss: 3.6970 - val_mae: 2.3103 - val_accuracy: 0.0000e+00
Epoch 17/100
17/17 [==============================] - 0s 4ms/step - loss: 3.1810 - mae: 2.5104 - accuracy: 0.0000e+00 - val_loss: 2.7768 - val_mae: 2.3131 - val_accuracy: 0.0000e+00
Epoch 18/100
17/17 [==============================] - 0s 4ms/step - loss: 2.0161 - mae: 2.5187 - accuracy: 0.0000e+00 - val_loss: 1.8144 - val_mae: 2.3159 - val_accuracy: 0.0000e+00
Epoch 19/100
17/17 [==============================] - 0s 3ms/step - loss: 1.1421 - mae: 2.5151 - accuracy: 0.0000e+00 - val_loss: 0.9175 - val_mae: 2.3183 - val_accuracy: 0.0000e+00
Epoch 20/100
17/17 [==============================] - 0s 4ms/step - loss: 0.0363 - mae: 2.5208 - accuracy: 0.0000e+00 - val_loss: 0.1350 - val_mae: 2.3205 - val_accuracy: 0.0000e+00
Epoch 21/100
17/17 [==============================] - 0s 4ms/step - loss: -0.9634 - mae: 2.5215 - accuracy: 0.0000e+00 - val_loss: -0.5890 - val_mae: 2.3216 - val_accuracy: 0.0000e+00
Epoch 22/100
17/17 [==============================] - 0s 4ms/step - loss: -1.6298 - mae: 2.5228 - accuracy: 0.0000e+00 - val_loss: -1.0732 - val_mae: 2.3223 - val_accuracy: 0.0000e+00
Epoch 23/100
17/17 [==============================] - 0s 4ms/step - loss: -2.2286 - mae: 2.5261 - accuracy: 0.0000e+00 - val_loss: -1.3935 - val_mae: 2.3229 - val_accuracy: 0.0000e+00
Epoch 24/100
17/17 [==============================] - 0s 4ms/step - loss: -2.7391 - mae: 2.5257 - accuracy: 0.0000e+00 - val_loss: -2.0796 - val_mae: 2.3236 - val_accuracy: 0.0000e+00
Epoch 25/100
17/17 [==============================] - 0s 6ms/step - loss: -3.1726 - mae: 2.5258 - accuracy: 0.0000e+00 - val_loss: -2.7100 - val_mae: 2.3242 - val_accuracy: 0.0000e+00
Epoch 26/100
17/17 [==============================] - 0s 4ms/step - loss: -3.5435 - mae: 2.5241 - accuracy: 0.0000e+00 - val_loss: -3.1445 - val_mae: 2.3248 - val_accuracy: 0.0000e+00
Epoch 27/100
17/17 [==============================] - 0s 4ms/step - loss: -4.1473 - mae: 2.5309 - accuracy: 0.0000e+00 - val_loss: -3.3916 - val_mae: 2.3254 - val_accuracy: 0.0000e+00
Epoch 28/100
17/17 [==============================] - 0s 6ms/step - loss: -4.5222 - mae: 2.5289 - accuracy: 0.0000e+00 - val_loss: -3.4078 - val_mae: 2.3254 - val_accuracy: 0.0000e+00
Epoch 29/100
17/17 [==============================] - 0s 5ms/step - loss: -4.5934 - mae: 2.5313 - accuracy: 0.0000e+00 - val_loss: -3.4510 - val_mae: 2.3254 - val_accuracy: 0.0000e+00
Epoch 30/100
17/17 [==============================] - 0s 4ms/step - loss: -4.4666 - mae: 2.5261 - accuracy: 0.0000e+00 - val_loss: -3.4013 - val_mae: 2.3255 - val_accuracy: 0.0000e+00
Epoch 31/100
17/17 [==============================] - 0s 4ms/step - loss: -4.4969 - mae: 2.5286 - accuracy: 0.0000e+00 - val_loss: -3.6364 - val_mae: 2.3254 - val_accuracy: 0.0000e+00
Epoch 32/100
17/17 [==============================] - 0s 4ms/step - loss: -4.4613 - mae: 2.5286 - accuracy: 0.0000e+00 - val_loss: -3.4672 - val_mae: 2.3254 - val_accuracy: 0.0000e+00
Epoch 33/100
17/17 [==============================] - 0s 4ms/step - loss: -4.4839 - mae: 2.5288 - accuracy: 0.0000e+00 - val_loss: -3.6557 - val_mae: 2.3254 - val_accuracy: 0.0000e+00
Epoch 34/100
17/17 [==============================] - 0s 6ms/step - loss: -4.5917 - mae: 2.5309 - accuracy: 0.0000e+00 - val_loss: -3.5606 - val_mae: 2.3254 - val_accuracy: 0.0000e+00
Epoch 35/100
17/17 [==============================] - 0s 5ms/step - loss: -4.4238 - mae: 2.5260 - accuracy: 0.0000e+00 - val_loss: -3.6552 - val_mae: 2.3253 - val_accuracy: 0.0000e+00
Epoch 36/100
17/17 [==============================] - 0s 5ms/step - loss: -4.4447 - mae: 2.5257 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3253 - val_accuracy: 0.0000e+00
Epoch 37/100
17/17 [==============================] - 0s 4ms/step - loss: -4.4976 - mae: 2.5280 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3253 - val_accuracy: 0.0000e+00
Epoch 38/100
17/17 [==============================] - 0s 4ms/step - loss: -4.5840 - mae: 2.5291 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3253 - val_accuracy: 0.0000e+00
Epoch 39/100
17/17 [==============================] - 0s 4ms/step - loss: -4.4384 - mae: 2.5268 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3252 - val_accuracy: 0.0000e+00
Epoch 40/100
17/17 [==============================] - 0s 4ms/step - loss: -4.4387 - mae: 2.5267 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3252 - val_accuracy: 0.0000e+00
Epoch 41/100
17/17 [==============================] - 0s 4ms/step - loss: -4.4095 - mae: 2.5243 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3252 - val_accuracy: 0.0000e+00
Epoch 42/100
17/17 [==============================] - 0s 3ms/step - loss: -4.4817 - mae: 2.5266 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3251 - val_accuracy: 0.0000e+00
Epoch 43/100
17/17 [==============================] - 0s 4ms/step - loss: -4.4434 - mae: 2.5250 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3251 - val_accuracy: 0.0000e+00
Epoch 44/100
17/17 [==============================] - 0s 4ms/step - loss: -4.2355 - mae: 2.5210 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3250 - val_accuracy: 0.0000e+00
Epoch 45/100
17/17 [==============================] - 0s 4ms/step - loss: -4.4462 - mae: 2.5248 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3250 - val_accuracy: 0.0000e+00
Epoch 46/100
17/17 [==============================] - 0s 4ms/step - loss: -4.4937 - mae: 2.5285 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3250 - val_accuracy: 0.0000e+00
Epoch 47/100
17/17 [==============================] - 0s 4ms/step - loss: -4.5392 - mae: 2.5289 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3250 - val_accuracy: 0.0000e+00
Epoch 48/100
17/17 [==============================] - 0s 5ms/step - loss: -4.4782 - mae: 2.5269 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3249 - val_accuracy: 0.0000e+00
Epoch 49/100
17/17 [==============================] - 0s 4ms/step - loss: -4.4217 - mae: 2.5240 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3249 - val_accuracy: 0.0000e+00
Epoch 50/100
17/17 [==============================] - 0s 4ms/step - loss: -4.5188 - mae: 2.5276 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3249 - val_accuracy: 0.0000e+00
Epoch 51/100
17/17 [==============================] - 0s 4ms/step - loss: -4.5914 - mae: 2.5280 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3248 - val_accuracy: 0.0000e+00
Epoch 52/100
17/17 [==============================] - 0s 4ms/step - loss: -4.4291 - mae: 2.5251 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3248 - val_accuracy: 0.0000e+00
Epoch 53/100
17/17 [==============================] - 0s 4ms/step - loss: -4.4513 - mae: 2.5262 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3248 - val_accuracy: 0.0000e+00
Epoch 54/100
17/17 [==============================] - 0s 4ms/step - loss: -4.4026 - mae: 2.5255 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3247 - val_accuracy: 0.0000e+00
Epoch 55/100
17/17 [==============================] - 0s 4ms/step - loss: -4.4794 - mae: 2.5275 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3247 - val_accuracy: 0.0000e+00
Epoch 56/100
17/17 [==============================] - 0s 4ms/step - loss: -4.6031 - mae: 2.5306 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3247 - val_accuracy: 0.0000e+00
Epoch 57/100
17/17 [==============================] - 0s 4ms/step - loss: -4.3214 - mae: 2.5223 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3246 - val_accuracy: 0.0000e+00
Epoch 58/100
17/17 [==============================] - 0s 4ms/step - loss: -4.4316 - mae: 2.5238 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3246 - val_accuracy: 0.0000e+00
Epoch 59/100
17/17 [==============================] - 0s 4ms/step - loss: -4.5925 - mae: 2.5285 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3246 - val_accuracy: 0.0000e+00
Epoch 60/100
17/17 [==============================] - 0s 4ms/step - loss: -4.5084 - mae: 2.5273 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3245 - val_accuracy: 0.0000e+00
Epoch 61/100
17/17 [==============================] - 0s 4ms/step - loss: -4.5451 - mae: 2.5269 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3245 - val_accuracy: 0.0000e+00
Epoch 62/100
17/17 [==============================] - 0s 4ms/step - loss: -4.4552 - mae: 2.5248 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3245 - val_accuracy: 0.0000e+00
Epoch 63/100
17/17 [==============================] - 0s 4ms/step - loss: -4.4916 - mae: 2.5259 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3244 - val_accuracy: 0.0000e+00
Epoch 64/100
17/17 [==============================] - 0s 5ms/step - loss: -4.4805 - mae: 2.5251 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3244 - val_accuracy: 0.0000e+00
Epoch 65/100
17/17 [==============================] - 0s 5ms/step - loss: -4.6268 - mae: 2.5294 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3244 - val_accuracy: 0.0000e+00
Epoch 66/100
17/17 [==============================] - 0s 5ms/step - loss: -4.4414 - mae: 2.5250 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3243 - val_accuracy: 0.0000e+00
Epoch 67/100
17/17 [==============================] - 0s 5ms/step - loss: -4.5781 - mae: 2.5275 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3243 - val_accuracy: 0.0000e+00
Epoch 68/100
17/17 [==============================] - 0s 4ms/step - loss: -4.5457 - mae: 2.5279 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3243 - val_accuracy: 0.0000e+00
Epoch 69/100
17/17 [==============================] - 0s 5ms/step - loss: -4.4725 - mae: 2.5246 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3242 - val_accuracy: 0.0000e+00
Epoch 70/100
17/17 [==============================] - 0s 4ms/step - loss: -4.4543 - mae: 2.5236 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3242 - val_accuracy: 0.0000e+00
Epoch 71/100
17/17 [==============================] - 0s 4ms/step - loss: -4.4883 - mae: 2.5254 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3241 - val_accuracy: 0.0000e+00
Epoch 72/100
17/17 [==============================] - 0s 4ms/step - loss: -4.5511 - mae: 2.5265 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3241 - val_accuracy: 0.0000e+00
Epoch 73/100
17/17 [==============================] - 0s 5ms/step - loss: -4.5925 - mae: 2.5281 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3241 - val_accuracy: 0.0000e+00
Epoch 74/100
17/17 [==============================] - 0s 4ms/step - loss: -4.4217 - mae: 2.5223 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3240 - val_accuracy: 0.0000e+00
Epoch 75/100
17/17 [==============================] - 0s 5ms/step - loss: -4.4099 - mae: 2.5207 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3240 - val_accuracy: 0.0000e+00
Epoch 76/100
17/17 [==============================] - 0s 4ms/step - loss: -4.4323 - mae: 2.5224 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3239 - val_accuracy: 0.0000e+00
Epoch 77/100
17/17 [==============================] - 0s 4ms/step - loss: -4.5303 - mae: 2.5258 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3239 - val_accuracy: 0.0000e+00
Epoch 78/100
17/17 [==============================] - 0s 4ms/step - loss: -4.3928 - mae: 2.5202 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3238 - val_accuracy: 0.0000e+00
Epoch 79/100
17/17 [==============================] - 0s 5ms/step - loss: -4.5268 - mae: 2.5238 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3238 - val_accuracy: 0.0000e+00
Epoch 80/100
17/17 [==============================] - 0s 4ms/step - loss: -4.4799 - mae: 2.5240 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3237 - val_accuracy: 0.0000e+00
Epoch 81/100
17/17 [==============================] - 0s 4ms/step - loss: -4.5491 - mae: 2.5260 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3237 - val_accuracy: 0.0000e+00
Epoch 82/100
17/17 [==============================] - 0s 5ms/step - loss: -4.5622 - mae: 2.5272 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3236 - val_accuracy: 0.0000e+00
Epoch 83/100
17/17 [==============================] - 0s 5ms/step - loss: -4.4865 - mae: 2.5232 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3236 - val_accuracy: 0.0000e+00
Epoch 84/100
17/17 [==============================] - 0s 5ms/step - loss: -4.5589 - mae: 2.5267 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3236 - val_accuracy: 0.0000e+00
Epoch 85/100
17/17 [==============================] - 0s 4ms/step - loss: -4.5464 - mae: 2.5256 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3235 - val_accuracy: 0.0000e+00
Epoch 86/100
17/17 [==============================] - 0s 5ms/step - loss: -4.5924 - mae: 2.5275 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3235 - val_accuracy: 0.0000e+00
Epoch 87/100
17/17 [==============================] - 0s 4ms/step - loss: -4.5796 - mae: 2.5251 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3235 - val_accuracy: 0.0000e+00
Epoch 88/100
17/17 [==============================] - 0s 4ms/step - loss: -4.5413 - mae: 2.5254 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3234 - val_accuracy: 0.0000e+00
Epoch 89/100
17/17 [==============================] - 0s 4ms/step - loss: -4.6143 - mae: 2.5266 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3234 - val_accuracy: 0.0000e+00
Epoch 90/100
17/17 [==============================] - 0s 5ms/step - loss: -4.6117 - mae: 2.5274 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3234 - val_accuracy: 0.0000e+00
Epoch 91/100
17/17 [==============================] - 0s 4ms/step - loss: -4.4731 - mae: 2.5228 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3233 - val_accuracy: 0.0000e+00
Epoch 92/100
17/17 [==============================] - 0s 4ms/step - loss: -4.5588 - mae: 2.5258 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3233 - val_accuracy: 0.0000e+00
Epoch 93/100
17/17 [==============================] - 0s 4ms/step - loss: -4.5822 - mae: 2.5255 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3232 - val_accuracy: 0.0000e+00
Epoch 94/100
17/17 [==============================] - 0s 4ms/step - loss: -4.4890 - mae: 2.5215 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3232 - val_accuracy: 0.0000e+00
Epoch 95/100
17/17 [==============================] - 0s 4ms/step - loss: -4.5709 - mae: 2.5269 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3231 - val_accuracy: 0.0000e+00
```

```
Epoch 96/100
17/17 [==============================] - 0s 5ms/step - loss: -4.6278 - mae: 2.5264 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3231 - val_accuracy: 0.0000e+00
Epoch 97/100
17/17 [==============================] - 0s 4ms/step - loss: -4.5016 - mae: 2.5227 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3230 - val_accuracy: 0.0000e+00
Epoch 98/100
17/17 [==============================] - 0s 4ms/step - loss: -4.5471 - mae: 2.5243 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3230 - val_accuracy: 0.0000e+00
Epoch 99/100
17/17 [==============================] - 0s 3ms/step - loss: -4.4532 - mae: 2.5222 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3229 - val_accuracy: 0.0000e+00
Epoch 100/100
17/17 [==============================] - 0s 4ms/step - loss: -4.6221 - mae: 2.5246 - accuracy: 0.0000e+00 - val_loss: -3.7957 - val_mae: 2.3229 - val_accuracy: 0.0000e+00
```

In [66]:
```
# оценим модель
scores = model.evaluate(x_test, y_test)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
9/9 [==============================] - 0s 4ms/step - loss: -4.3965 - mae: 2.4469 - accuracy: 0.0000e+00

mae: 244.69%
```

In [67]:
```
# Посмотрим на потери модели

model_hist.history
```

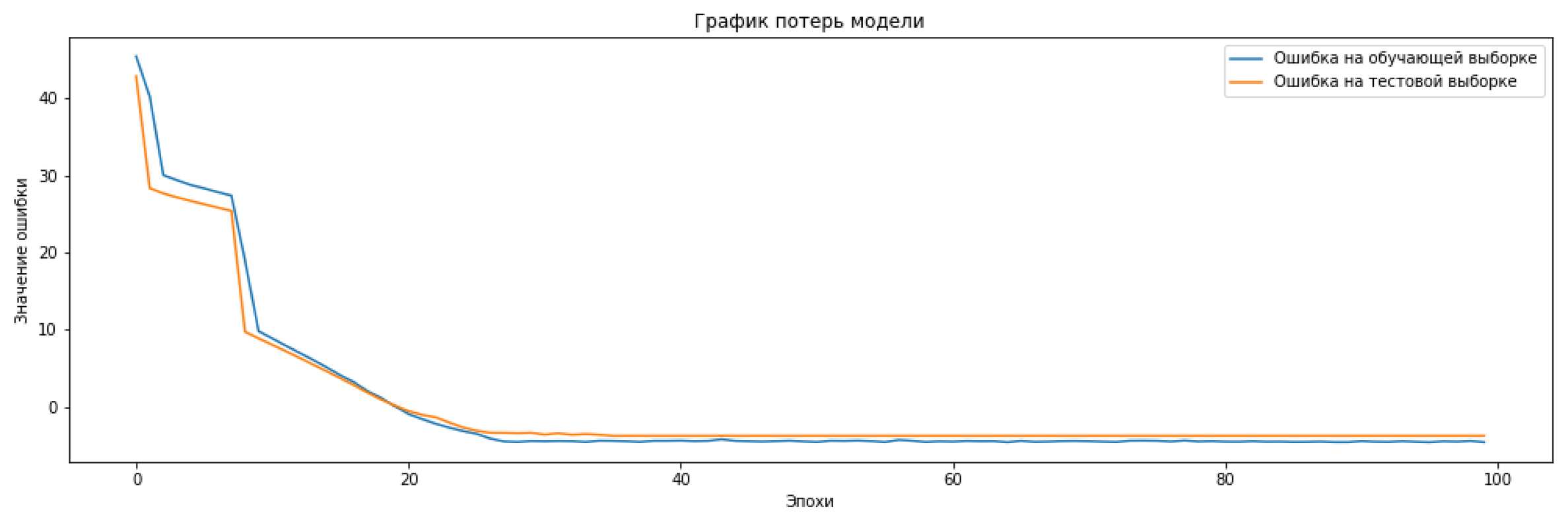
Out[67]: {'loss': [45.44175720214844,
40.21364212036133,
30.000333786010742,
29.356956481933594,
28.741199493408203,
28.298103332519531,
27.800493240356445,
27.354223251342773,
18.955575942993164,
9.789545059204102,
8.862156867980957,
7.907751083374023,
6.989405155181885,
6.074981212615967,
5.101145267486572,
4.080352783203125,
3.1809909343719482,
2.016057014465332,
1.1421365737915004,
0.03628274053335519,
-0.9634304644267307,
-1.6298295259475708,
-2.228590726852417,
-2.7391483783721924,
-3.1725752353668213,
-3.543485641479492,
-4.147338867187,
-4.522206783294678,
-4.593357563018799,
-4.466593742370605,
-4.496935367584228,
-4.461312770843506,
-4.483090606933594,
-4.591719627380371,
-4.423764705657959,
-4.444697380065918,
-4.497642004252055,
-4.584041595458984,
-4.438376426696777,
-4.438713550567627,
-4.409480094909668,
-4.481719970703125,
-4.443446159362793,
-4.235463619232178,
-4.446217536926269,
-4.493743896484375,
-4.539159774780273,
-4.478187084197998,
-4.421745777130127,
-4.518826007843018,
-4.591385364532471,
-4.429142475128174,
-4.451309680938721,
-4.402649879455566,
-4.479367713001709,
-4.603068828582764,
-4.321437358856201,
-4.431607723236084,
-4.592545509338379,
-4.508388042449951,
-4.545135021209717,
-4.455235004425049,
-4.491605281829834,
-4.488539798736572,
-4.626796722412109,
-4.441429615020752,
-4.578063964843375,
-4.545735836029053,
-4.472460746765137,
-4.454301357269287,
-4.488348484039307,
-4.551143646240234,
-4.592499732971191,
-4.421683311462402,
-4.40994930267334,
-4.432267189025879,
-4.530339241027832,
-4.392793655395508,
-4.526751041412353 5,
-4.479893207550049,
-4.549111366271973,
-4.562169551849365,
-4.486512660980225,
-4.558922290802002,
-4.546354293823242,
-4.592384338378906,
-4.579616546630859,
-4.541306018829346,
-4.61434793472229,
-4.611684322357178,
-4.473118019074609,
-4.558835506439209,
-4.582152366638184,
-4.488958025252059,
-4.570934299654297,
-4.627841949462891,
-4.501616954803467,
-4.547112464904785,
-4.453245639801025,
-4.622082233428955],
'mae': [3.3252146244049072,
3.224177837371826,
3.0046938311311035,
2.9674465656280518,
2.9549517631530 76,
2.9462203979492188,
2.949082136154175,
2.9540345668792725,
2.7619240828966064,
2.5313186645507812,
2.514868736267 09,
2.5100555419921875,
2.508671998977661,
2.5058815479278564,
2.508711099624634,
2.5125389099121094,
2.5103752613067627,
2.5187108516693115,
2.5151169300079346,
2.520775794982 91,
2.5214872360229 49,
2.5228374004364014,
2.5261449813842773,
2.5257291793823 24,
2.525834798812866,
2.5241098403930664,
2.5309236049652 1,
2.5289487838745117,
2.5313003063201904,
2.5260084761810303,
2.5285565853118896,
2.5285623037357788,
2.5287876129150 39,
2.5309453010559 08,
2.5259850025177,
2.5257163047790527,
2.5280175209045 41,
2.5291383266448975,
2.5268409252166 75,
2.5267488956451416,
2.5243330001831055,
2.5266313552856445,
2.5250384880586 67,
2.5209510326 3855,
2.5248157977005786,
2.5284876823425293,
2.5288527011871 34,
2.5268047942352295,
2.5239574909210205,
2.527587890625,
2.5279967784881 59,
2.5250740051269 53,
2.5262198448181152,
2.5255118263824463,
2.5274834632873535,
2.5305755138397217,
2.5222625732421875,
2.5237896442413 33,
2.5285477638244 63,
2.5272965431213 38,
2.5269236564636 23,
2.5248186588287354,
2.5259342193603516,
2.5250597000122 07,
2.5293927192687 99,
2.5250437259674072,
2.5274734497070312,
2.5278713703155518,
2.5245721340179443,
2.5236456394195557,
2.5254068374633 79,
2.5264739990234375,
2.5280909538269043,
2.5223488807678223,
2.5207130908966064,
2.5223610401153564,
2.5258219242095947,
2.5281698989868 93,
2.5237748622894287,
2.5240201950073 24,
2.5260886859130 86,
2.5272412300109863,
2.5231523513793945,
2.5266885757446289,
2.5256383419036865,
2.5275177955627 44,
2.5251474380493164,
2.5253968238830566,
2.5265769958496094,
2.5273656845092773,

2.522803783416748,
2.5257554054260254,
2.5255179405212402,
2.52152156829834,
2.5269033908843994,
2.5263683795928955,
2.5226712226867676,
2.524304151535034,
2.5222179889678955,
2.5245871543884277],
'accuracy': [0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0],
'val_loss': [42.86212158203125,
28.31760025024414,
27.63954734802246,
27.126041412353516,
26.67052459716797,
26.23000717163086,
25.800155639648438,
25.37271499633789,
9.720048904418945,
8.856014251708984,
8.030655860900879,
7.188610553741455,
6.341849327087402,
5.465333461761475,
4.591294288635254,
3.69698429107666,
2.776841878890991,
1.8143974542617798,
0.9174960255622864,
0.1349557340145111,
-0.5890403985977173,
-1.073218584060669,
-1.3935233354568481,
-2.0796055793762207,
-2.710003137588501,
-3.144476890563965,
-3.3915717601776123,
-3.4077975749969482,
-3.451042652130127,
-3.4012913703918457,
-3.6364450454711914,
-3.4671623706817627,
-3.655653476715088,
-3.5606372356414795,
-3.655190944671631,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,
-3.7957117557525635,

```
 -3.7957117557525635,
 -3.7957117557525635,
 -3.7957117557525635,
 -3.7957117557525635,
 -3.7957117557525635,
 -3.7957117557525635,
 -3.7957117557525635,
 -3.7957117557525635,
 -3.7957117557525635,
 -3.7957117557525635,
 -3.7957117557525635,
 -3.7957117557525635,
 -3.7957117557525635,
 -3.7957117557525635,
 -3.7957117557525635,
 -3.7957117557525635,
 -3.7957117557525635,
 -3.7957117557525635,
 -3.7957117557525635,
 -3.7957117557525635],
'val_mae': [3.111691474914551,
 2.8310041427612305,
 2.776968240737915,
 2.7562315464019775,
 2.7449276447296143,
 2.745009183883667,
 2.7483766078948975,
 2.754690408706665,
 2.338858127593994,
 2.3175806999206543,
 2.30784273147583,
 2.304110527038574,
 2.3036677837371826,
 2.305101156234741,
 2.3074662685394287,
 2.310274839401245,
 2.3130571842193604,
 2.3159217834472656,
 2.318300724029541,
 2.3204519748687744,
 2.3216309547424316,
 2.3222975730895996,
 2.322934627532959,
 2.3235762119293213,
 2.3242135047912598,
 2.3248116970062256,
 2.3253586292266846,
 2.325399398803711,
 2.3254246711730957,
 2.3254504203796387,
 2.325435161590576,
 2.325409412384033,
 2.3253846168518066,
 2.3253674507141113,
 2.3253350257873535,
 2.3253066539764404,
 2.3252811431884766,
 2.3252623081207275,
 2.325230121612549,
 2.325197219848633,
 2.3251590728759766,
 2.325129508972168,
 2.32509708404541,
 2.325047016143799,
 2.3250138759613037,
 2.3249850273132324,
 2.324960231781006,
 2.3249311447143555,
 2.32489013671875,
 2.324862480163574,
 2.324836254119873,
 2.324798107147217,
 2.324763774871826,
 2.3247218132019043,
 2.324693202972412,
 2.324673652648926,
 2.3246231079101562,
 2.3245866298675537,
 2.3245649337768555,
 2.324533700942993,
 2.3245058059692383,
 2.324456214904785,
 2.324422597885132,
 2.3243844509124756,
 2.3243627548217773,
 2.3243136405944824,
 2.3242878913879395,
 2.324258804321289,
 2.324219226837158,
 2.324166774749756,
 2.3241186141967773,
 2.3240861892700195,
 2.3240580558776855,
 2.32401180267334,
 2.3239660263061523,
 2.323916435241699,
 2.323878288269043,
 2.323812484741211,
 2.3237781524658203,
 2.323730707168579,
 2.3236942291259766,
 2.3236496448516846,
 2.323605537414551,
 2.3235678672790527,
 2.3235199451446533,
 2.323488712310791,
 2.323452949523926,
 2.323415517807007,
 2.3233842849731445,
 2.3233518600463867,
 2.323300361633301,
 2.323258399963379,
 2.323223114013672,
 2.3231778144836426,
 2.3231210708618164,
 2.323089122772217,
 2.323037624359131,
 2.3229904174804688,
 2.3229269981384277,
 2.3228919506073],
'val_accuracy': [0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
```

```
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0]}
```

In [68]:
```
# Посмотрим на график потерь на тренировочной и тестовой выборках
def model_loss_plot(model_hist):
    plt.figure(figsize = (17,5))
    plt.plot(model_hist.history['loss'],
             label = 'ошибка на обучающей выборке')
    plt.plot(model_hist.history['val_loss'],
            label = 'ошибка на тестовой выборке')
    plt.title('График потерь модели')
    plt.ylabel('Значение ошибки')
    plt.xlabel('Эпохи')
    plt.legend(['Ошибка на обучающей выборке', 'Ошибка на тестовой выборке'], loc='best')
    plt.show()
model_loss_plot(model_hist)
```



In [69]:
```
# Зададим функцию для визуализации факт/прогноз для результатов моделей
# Посмотрим на график результата работы модели
def actual_and_predicted_plot(orig, predict, var, model_name):
    plt.figure(figsize=(17,5))
    plt.title(f'Тестовые и прогнозные значения: {model_name}')
    plt.plot(orig, label = 'Тест')
    plt.plot(predict, label = 'Прогноз')
    plt.legend(loc = 'best')
    plt.ylabel(var)
    plt.xlabel('Количество наблюдений')
    plt.show()
actual_and_predicted_plot(y_test.values, model.predict(x_test.values), 'Соотношение матрица/наполнитель', 'Keras_neuronet')
```

```
9/9 [==============================] - 0s 2ms/step
```

In [73]:
```
# Сконфигурируем другую модель, зададим слои
from tensorflow.keras.layers import Dense
model1 = tf.keras.Sequential([x_train_n, layers.Dense(128, activation='relu'),
                                                layers.Dense(128, activation='relu'),
                                                layers.Dense(128, activation='relu'),
                                                layers.Dense(64, activation='relu'),
                                                layers.Dense(64, activation='relu'),
                                                layers.Dense(32, activation='relu'),
                                                layers.Dense(16, activation='relu'),
                                                layers.Dense(1)
                            ])

model1.compile(optimizer = tf.keras.optimizers.Adam(0.001), loss = 'mean_squared_error', metrics = [tf.keras.metrics.RootMeanSquaredError()])
# Посмотрим на архитектуру модели

model1.summary()
```

```
Model: "sequential_196"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization (Normalizatio  (None, 14)               29
 n)

 dense_498 (Dense)           (None, 128)               1920

 dense_499 (Dense)           (None, 128)               16512

 dense_500 (Dense)           (None, 128)               16512

 dense_501 (Dense)           (None, 64)                8256

 dense_502 (Dense)           (None, 64)                4160

 dense_503 (Dense)           (None, 32)                2080

 dense_504 (Dense)           (None, 16)                528

 dense_505 (Dense)           (None, 1)                 17

=================================================================
Total params: 50,014
Trainable params: 49,985
Non-trainable params: 29
_________________________________________________________________
```

In [74]:
```
# Обучим модель

model_hist1 = model1.fit(
    x_train,
    y_train,
    epochs = 100,
    verbose = 1,
    validation_split = 0.2)
```

```
Epoch 1/100
17/17 [==============================] - 2s 24ms/step - loss: 6.9116 - root_mean_squared_error: 2.6290 - val_loss: 1.3523 - val_root_mean_squared_error: 1.1629
Epoch 2/100
17/17 [==============================] - 0s 5ms/step - loss: 1.4837 - root_mean_squared_error: 1.2181 - val_loss: 1.3695 - val_root_mean_squared_error: 1.1703
Epoch 3/100
17/17 [==============================] - 0s 5ms/step - loss: 1.1125 - root_mean_squared_error: 1.0547 - val_loss: 1.1390 - val_root_mean_squared_error: 1.0672
Epoch 4/100
17/17 [==============================] - 0s 5ms/step - loss: 0.9776 - root_mean_squared_error: 0.9888 - val_loss: 1.4329 - val_root_mean_squared_error: 1.1970
Epoch 5/100
17/17 [==============================] - 0s 4ms/step - loss: 0.8868 - root_mean_squared_error: 0.9417 - val_loss: 1.2451 - val_root_mean_squared_error: 1.1158
Epoch 6/100
17/17 [==============================] - 0s 5ms/step - loss: 0.8349 - root_mean_squared_error: 0.9137 - val_loss: 1.1285 - val_root_mean_squared_error: 1.0623
Epoch 7/100
17/17 [==============================] - 0s 5ms/step - loss: 0.7899 - root_mean_squared_error: 0.8888 - val_loss: 1.1436 - val_root_mean_squared_error: 1.0694
Epoch 8/100
17/17 [==============================] - 0s 5ms/step - loss: 0.7581 - root_mean_squared_error: 0.8707 - val_loss: 1.1829 - val_root_mean_squared_error: 1.0876
Epoch 9/100
17/17 [==============================] - 0s 5ms/step - loss: 0.7434 - root_mean_squared_error: 0.8622 - val_loss: 1.1236 - val_root_mean_squared_error: 1.0600
Epoch 10/100
17/17 [==============================] - 0s 7ms/step - loss: 0.6999 - root_mean_squared_error: 0.8366 - val_loss: 1.0859 - val_root_mean_squared_error: 1.0421
Epoch 11/100
17/17 [==============================] - 0s 6ms/step - loss: 0.6497 - root_mean_squared_error: 0.8060 - val_loss: 1.0566 - val_root_mean_squared_error: 1.0279
Epoch 12/100
17/17 [==============================] - 0s 5ms/step - loss: 0.6521 - root_mean_squared_error: 0.8075 - val_loss: 1.0468 - val_root_mean_squared_error: 1.0232
Epoch 13/100
17/17 [==============================] - 0s 5ms/step - loss: 0.5999 - root_mean_squared_error: 0.7746 - val_loss: 1.0707 - val_root_mean_squared_error: 1.0347
Epoch 14/100
17/17 [==============================] - 0s 6ms/step - loss: 0.5687 - root_mean_squared_error: 0.7541 - val_loss: 1.2484 - val_root_mean_squared_error: 1.1173
Epoch 15/100
17/17 [==============================] - 0s 6ms/step - loss: 0.5462 - root_mean_squared_error: 0.7390 - val_loss: 1.1708 - val_root_mean_squared_error: 1.0820
Epoch 16/100
17/17 [==============================] - 0s 5ms/step - loss: 0.4648 - root_mean_squared_error: 0.6818 - val_loss: 1.2609 - val_root_mean_squared_error: 1.1229
Epoch 17/100
17/17 [==============================] - 0s 5ms/step - loss: 0.4242 - root_mean_squared_error: 0.6513 - val_loss: 1.1172 - val_root_mean_squared_error: 1.0570
Epoch 18/100
17/17 [==============================] - 0s 5ms/step - loss: 0.4339 - root_mean_squared_error: 0.6587 - val_loss: 1.2005 - val_root_mean_squared_error: 1.0957
Epoch 19/100
17/17 [==============================] - 0s 6ms/step - loss: 0.3918 - root_mean_squared_error: 0.6259 - val_loss: 1.2835 - val_root_mean_squared_error: 1.1329
Epoch 20/100
17/17 [==============================] - 0s 4ms/step - loss: 0.3164 - root_mean_squared_error: 0.5625 - val_loss: 1.2995 - val_root_mean_squared_error: 1.1400
Epoch 21/100
17/17 [==============================] - 0s 5ms/step - loss: 0.2615 - root_mean_squared_error: 0.5113 - val_loss: 1.2745 - val_root_mean_squared_error: 1.1289
Epoch 22/100
17/17 [==============================] - 0s 5ms/step - loss: 0.2269 - root_mean_squared_error: 0.4763 - val_loss: 1.2452 - val_root_mean_squared_error: 1.1159
Epoch 23/100
17/17 [==============================] - 0s 6ms/step - loss: 0.2309 - root_mean_squared_error: 0.4806 - val_loss: 1.4742 - val_root_mean_squared_error: 1.2142
Epoch 24/100
17/17 [==============================] - 0s 5ms/step - loss: 0.2030 - root_mean_squared_error: 0.4506 - val_loss: 1.2805 - val_root_mean_squared_error: 1.1316
Epoch 25/100
17/17 [==============================] - 0s 5ms/step - loss: 0.1656 - root_mean_squared_error: 0.4070 - val_loss: 1.3946 - val_root_mean_squared_error: 1.1809
Epoch 26/100
17/17 [==============================] - 0s 5ms/step - loss: 0.1254 - root_mean_squared_error: 0.3541 - val_loss: 1.4239 - val_root_mean_squared_error: 1.1933
Epoch 27/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1145 - root_mean_squared_error: 0.3384 - val_loss: 1.4365 - val_root_mean_squared_error: 1.1985
Epoch 28/100
17/17 [==============================] - 0s 5ms/step - loss: 0.1228 - root_mean_squared_error: 0.3504 - val_loss: 1.4626 - val_root_mean_squared_error: 1.2094
Epoch 29/100
17/17 [==============================] - 0s 6ms/step - loss: 0.0951 - root_mean_squared_error: 0.3083 - val_loss: 1.6037 - val_root_mean_squared_error: 1.2664
Epoch 30/100
17/17 [==============================] - 0s 8ms/step - loss: 0.0855 - root_mean_squared_error: 0.2923 - val_loss: 1.5551 - val_root_mean_squared_error: 1.2470
Epoch 31/100
17/17 [==============================] - 0s 7ms/step - loss: 0.0772 - root_mean_squared_error: 0.2778 - val_loss: 1.6726 - val_root_mean_squared_error: 1.2933
Epoch 32/100
17/17 [==============================] - 0s 6ms/step - loss: 0.0986 - root_mean_squared_error: 0.3139 - val_loss: 1.4547 - val_root_mean_squared_error: 1.2061
Epoch 33/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0612 - root_mean_squared_error: 0.2473 - val_loss: 1.5642 - val_root_mean_squared_error: 1.2507
Epoch 34/100
17/17 [==============================] - 0s 6ms/step - loss: 0.0525 - root_mean_squared_error: 0.2292 - val_loss: 1.5199 - val_root_mean_squared_error: 1.2329
Epoch 35/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0716 - root_mean_squared_error: 0.2677 - val_loss: 1.3483 - val_root_mean_squared_error: 1.1612
Epoch 36/100
17/17 [==============================] - 0s 7ms/step - loss: 0.0802 - root_mean_squared_error: 0.2833 - val_loss: 1.5039 - val_root_mean_squared_error: 1.2263
Epoch 37/100
17/17 [==============================] - 0s 7ms/step - loss: 0.0471 - root_mean_squared_error: 0.2171 - val_loss: 1.6250 - val_root_mean_squared_error: 1.2748
Epoch 38/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0353 - root_mean_squared_error: 0.1878 - val_loss: 1.4969 - val_root_mean_squared_error: 1.2235
Epoch 39/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0211 - root_mean_squared_error: 0.1454 - val_loss: 1.5514 - val_root_mean_squared_error: 1.2456
Epoch 40/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0158 - root_mean_squared_error: 0.1257 - val_loss: 1.5651 - val_root_mean_squared_error: 1.2511
Epoch 41/100
17/17 [==============================] - 0s 4ms/step - loss: 0.0152 - root_mean_squared_error: 0.1234 - val_loss: 1.5103 - val_root_mean_squared_error: 1.2290
Epoch 42/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0122 - root_mean_squared_error: 0.1103 - val_loss: 1.5152 - val_root_mean_squared_error: 1.2309
Epoch 43/100
17/17 [==============================] - 0s 6ms/step - loss: 0.0182 - root_mean_squared_error: 0.1348 - val_loss: 1.4904 - val_root_mean_squared_error: 1.2208
Epoch 44/100
17/17 [==============================] - 0s 6ms/step - loss: 0.0170 - root_mean_squared_error: 0.1302 - val_loss: 1.5196 - val_root_mean_squared_error: 1.2327
Epoch 45/100
17/17 [==============================] - 0s 4ms/step - loss: 0.0119 - root_mean_squared_error: 0.1091 - val_loss: 1.5081 - val_root_mean_squared_error: 1.2280
Epoch 46/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0118 - root_mean_squared_error: 0.1088 - val_loss: 1.5411 - val_root_mean_squared_error: 1.2414
Epoch 47/100
17/17 [==============================] - 0s 6ms/step - loss: 0.0092 - root_mean_squared_error: 0.0961 - val_loss: 1.5360 - val_root_mean_squared_error: 1.2393
Epoch 48/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0055 - root_mean_squared_error: 0.0744 - val_loss: 1.5291 - val_root_mean_squared_error: 1.2366
Epoch 49/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0056 - root_mean_squared_error: 0.0750 - val_loss: 1.5413 - val_root_mean_squared_error: 1.2415
Epoch 50/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0087 - root_mean_squared_error: 0.0931 - val_loss: 1.5389 - val_root_mean_squared_error: 1.2405
Epoch 51/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0128 - root_mean_squared_error: 0.1130 - val_loss: 1.5877 - val_root_mean_squared_error: 1.2600
Epoch 52/100
17/17 [==============================] - 0s 6ms/step - loss: 0.0109 - root_mean_squared_error: 0.1043 - val_loss: 1.5732 - val_root_mean_squared_error: 1.2543
Epoch 53/100
17/17 [==============================] - 0s 4ms/step - loss: 0.0089 - root_mean_squared_error: 0.0944 - val_loss: 1.5175 - val_root_mean_squared_error: 1.2319
Epoch 54/100
17/17 [==============================] - 0s 4ms/step - loss: 0.0076 - root_mean_squared_error: 0.0874 - val_loss: 1.5326 - val_root_mean_squared_error: 1.2380
Epoch 55/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0076 - root_mean_squared_error: 0.0872 - val_loss: 1.5608 - val_root_mean_squared_error: 1.2493
Epoch 56/100
17/17 [==============================] - 0s 6ms/step - loss: 0.0099 - root_mean_squared_error: 0.0993 - val_loss: 1.5859 - val_root_mean_squared_error: 1.2593
Epoch 57/100
17/17 [==============================] - 0s 6ms/step - loss: 0.0074 - root_mean_squared_error: 0.0862 - val_loss: 1.5621 - val_root_mean_squared_error: 1.2499
Epoch 58/100
17/17 [==============================] - 0s 4ms/step - loss: 0.0075 - root_mean_squared_error: 0.0865 - val_loss: 1.5689 - val_root_mean_squared_error: 1.2526
Epoch 59/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0058 - root_mean_squared_error: 0.0763 - val_loss: 1.5171 - val_root_mean_squared_error: 1.2317
Epoch 60/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0053 - root_mean_squared_error: 0.0727 - val_loss: 1.5943 - val_root_mean_squared_error: 1.2627
Epoch 61/100
17/17 [==============================] - 0s 4ms/step - loss: 0.0051 - root_mean_squared_error: 0.0718 - val_loss: 1.5700 - val_root_mean_squared_error: 1.2530
Epoch 62/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0056 - root_mean_squared_error: 0.0751 - val_loss: 1.5749 - val_root_mean_squared_error: 1.2549
Epoch 63/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0049 - root_mean_squared_error: 0.0700 - val_loss: 1.5446 - val_root_mean_squared_error: 1.2428
Epoch 64/100
17/17 [==============================] - 0s 4ms/step - loss: 0.0044 - root_mean_squared_error: 0.0667 - val_loss: 1.5586 - val_root_mean_squared_error: 1.2484
Epoch 65/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0045 - root_mean_squared_error: 0.0673 - val_loss: 1.5272 - val_root_mean_squared_error: 1.2358
Epoch 66/100
17/17 [==============================] - 0s 8ms/step - loss: 0.0032 - root_mean_squared_error: 0.0570 - val_loss: 1.5267 - val_root_mean_squared_error: 1.2356
Epoch 67/100
17/17 [==============================] - 0s 4ms/step - loss: 0.0044 - root_mean_squared_error: 0.0662 - val_loss: 1.5345 - val_root_mean_squared_error: 1.2388
Epoch 68/100
17/17 [==============================] - 0s 4ms/step - loss: 0.0076 - root_mean_squared_error: 0.0872 - val_loss: 1.4946 - val_root_mean_squared_error: 1.2226
Epoch 69/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0083 - root_mean_squared_error: 0.0912 - val_loss: 1.5901 - val_root_mean_squared_error: 1.2610
Epoch 70/100
17/17 [==============================] - 0s 4ms/step - loss: 0.0066 - root_mean_squared_error: 0.0809 - val_loss: 1.5288 - val_root_mean_squared_error: 1.2364
Epoch 71/100
17/17 [==============================] - 0s 4ms/step - loss: 0.0037 - root_mean_squared_error: 0.0610 - val_loss: 1.5119 - val_root_mean_squared_error: 1.2296
Epoch 72/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0041 - root_mean_squared_error: 0.0638 - val_loss: 1.4940 - val_root_mean_squared_error: 1.2223
Epoch 73/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0041 - root_mean_squared_error: 0.0643 - val_loss: 1.5363 - val_root_mean_squared_error: 1.2395
Epoch 74/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0052 - root_mean_squared_error: 0.0720 - val_loss: 1.5375 - val_root_mean_squared_error: 1.2400
Epoch 75/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0048 - root_mean_squared_error: 0.0695 - val_loss: 1.5240 - val_root_mean_squared_error: 1.2345
Epoch 76/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0070 - root_mean_squared_error: 0.0838 - val_loss: 1.5013 - val_root_mean_squared_error: 1.2253
Epoch 77/100
17/17 [==============================] - 0s 4ms/step - loss: 0.0099 - root_mean_squared_error: 0.0996 - val_loss: 1.5502 - val_root_mean_squared_error: 1.2451
Epoch 78/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0171 - root_mean_squared_error: 0.1309 - val_loss: 1.5146 - val_root_mean_squared_error: 1.2307
Epoch 79/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0220 - root_mean_squared_error: 0.1483 - val_loss: 1.5284 - val_root_mean_squared_error: 1.2363
Epoch 80/100
17/17 [==============================] - 0s 3ms/step - loss: 0.0236 - root_mean_squared_error: 0.1537 - val_loss: 1.5162 - val_root_mean_squared_error: 1.2313
Epoch 81/100
17/17 [==============================] - 0s 4ms/step - loss: 0.0651 - root_mean_squared_error: 0.2551 - val_loss: 1.5771 - val_root_mean_squared_error: 1.2558
Epoch 82/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0634 - root_mean_squared_error: 0.2519 - val_loss: 1.4032 - val_root_mean_squared_error: 1.1846
Epoch 83/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0595 - root_mean_squared_error: 0.2440 - val_loss: 1.4747 - val_root_mean_squared_error: 1.2144
Epoch 84/100
17/17 [==============================] - 0s 6ms/step - loss: 0.0729 - root_mean_squared_error: 0.2700 - val_loss: 1.4150 - val_root_mean_squared_error: 1.1895
Epoch 85/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0766 - root_mean_squared_error: 0.2767 - val_loss: 1.6155 - val_root_mean_squared_error: 1.2710
Epoch 86/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0534 - root_mean_squared_error: 0.2311 - val_loss: 1.5398 - val_root_mean_squared_error: 1.2409
Epoch 87/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0289 - root_mean_squared_error: 0.1699 - val_loss: 1.4532 - val_root_mean_squared_error: 1.2055
Epoch 88/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0186 - root_mean_squared_error: 0.1363 - val_loss: 1.4613 - val_root_mean_squared_error: 1.2088
Epoch 89/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0122 - root_mean_squared_error: 0.1104 - val_loss: 1.5166 - val_root_mean_squared_error: 1.2315
Epoch 90/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0086 - root_mean_squared_error: 0.0925 - val_loss: 1.5206 - val_root_mean_squared_error: 1.2331
Epoch 91/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0074 - root_mean_squared_error: 0.0858 - val_loss: 1.5245 - val_root_mean_squared_error: 1.2347
Epoch 92/100
17/17 [==============================] - 0s 6ms/step - loss: 0.0049 - root_mean_squared_error: 0.0699 - val_loss: 1.5279 - val_root_mean_squared_error: 1.2361
Epoch 93/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0038 - root_mean_squared_error: 0.0618 - val_loss: 1.4940 - val_root_mean_squared_error: 1.2223
Epoch 94/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0022 - root_mean_squared_error: 0.0470 - val_loss: 1.5091 - val_root_mean_squared_error: 1.2284
Epoch 95/100
17/17 [==============================] - 0s 4ms/step - loss: 0.0015 - root_mean_squared_error: 0.0383 - val_loss: 1.5007 - val_root_mean_squared_error: 1.2250
```

```
Epoch 96/100
17/17 [==============================] - 0s 4ms/step - loss: 9.5466e-04 - root_mean_squared_error: 0.0309 - val_loss: 1.4951 - val_root_mean_squared_error: 1.2227
Epoch 97/100
17/17 [==============================] - 0s 5ms/step - loss: 7.6219e-04 - root_mean_squared_error: 0.0276 - val_loss: 1.4798 - val_root_mean_squared_error: 1.2165
Epoch 98/100
17/17 [==============================] - 0s 5ms/step - loss: 8.5690e-04 - root_mean_squared_error: 0.0293 - val_loss: 1.4941 - val_root_mean_squared_error: 1.2223
Epoch 99/100
17/17 [==============================] - 0s 5ms/step - loss: 7.1959e-04 - root_mean_squared_error: 0.0268 - val_loss: 1.4963 - val_root_mean_squared_error: 1.2232
Epoch 100/100
17/17 [==============================] - 0s 5ms/step - loss: 4.2799e-04 - root_mean_squared_error: 0.0207 - val_loss: 1.4880 - val_root_mean_squared_error: 1.2198
```

In [75]:
```
model1.evaluate(x_test, y_test)
```

```
9/9 [==============================] - 0s 2ms/step - loss: 1.2427 - root_mean_squared_error: 1.1147
[1.2426555156707764, 1.1147445440292358]
```

Out[75]:

In [76]:
```
y_pred_model = model1.predict(x_test)

print('Model Results:')
print('Model_MAE: ', round(mean_absolute_error(y_test, y_pred_model)))
print('Model_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test, y_pred_model)))
print("Test score: {:.2f}".format(mean_squared_error(y_test, y_pred_model)))
```

```
9/9 [==============================] - 0s 2ms/step
Model Results:
Model_MAE:  1
Model_MAPE: 0.37
Test score: 1.24
```

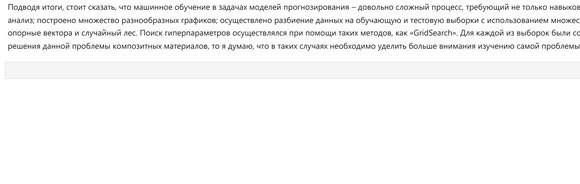
In [77]:
```
# Посмотрим на потери модели

model_hist1.history
```

Out[77]: {'loss': [6.911602497100 83,
1.4837313890457153,
1.1124696731567383,
0.9776344299316406,
0.8868167400360107,
0.8349333405494 69,
0.7899431586265564,
0.7581437826156616,
0.7434050440788269,
0.6998098983001709,
0.6496738791465759,
0.6520502567291 26,
0.5999318957328796,
0.5686761140823364,
0.5461846590042114,
0.4648489654064 1785,
0.4241849482059479,
0.4338574409484 8633,
0.3917693793773651,
0.3163940012454 9866,
0.2614695727825165,
0.2269036769866 9434,
0.2309314012527 4658,
0.2030475884675 9796,
0.1656425744295 1202,
0.1253712624311 4471,
0.1145192906268 4904,
0.1228044852614 4028,
0.0959060169696 80786,
0.0854658186435 6995,
0.0771598887418 1639,
0.0985619202256 2027,
0.0611545890569 6869,
0.0525181666016 5787,
0.0716446340084 0759,
0.0802436470985 4126,
0.0471146479240 80055,
0.0352731645107 2693,
0.0211301576346 15898,
0.0157913900911 808,
0.0155220147557 55663,
0.0121692223474 38335,
0.0181041924786 5677,
0.0169633403420 4483,
0.0119011048227 5486,
0.0118269277736 54461,
0.0092321578413 24806,
0.0055389720946 55037,
0.0056303199380 636215,
0.0086750313639 6408,
0.0127749461680 65871,
0.0108849173411 72695,
0.0089181801304 22115,
0.0076458333060 14538,
0.0075908131181 10844,
0.0098677324131 13117,
0.0074233766645 1931,
0.0074796383030 967,
0.0058201323263 34715,
0.0052796821109 95054,
0.0051492604765 32936,
0.0056469715200 36459,
0.0049045523628 59249,
0.0044467193074 5244,
0.0045305322855 71098,
0.0032483388204 12755,
0.0043771746568 38179,
0.0076099932193 7561,
0.0083109885454 17786,
0.0065505807287 99185,
0.0037173980381 33979,
0.0040707373991 60862,
0.0041280658915 63892,
0.0051895808428 5925,
0.0048302481882 27415,
0.0070278355851 76945,
0.0099262762814 7602,
0.0171263217926 0254,
0.0219981055706 73943,
0.0236144568771 1239,
0.0650839209556 5796,
0.0634452849626 5411,
0.0595439597964 2868,
0.0728964060054 49677,
0.0765846446155 6018,
0.0534144230186 93924,
0.0288715753704 30946,
0.0185721050947 9046,
0.0121836559846 99726,
0.0085530597165 699,
0.0073542851023 37599,
0.0048905122093 856335,
0.0038177852984 517813,
0.0022083872463 54144,
0.0014640226727 351546,
0.0009546556393 615901,
0.0007621912518 516183,
0.0008568979683 332145,
0.0007195918587 967753,
0.0004279864951 968193],
'root_mean_squared_error': [2.628992795944214,
1.2180851697921753,
1.0547367334365845,
0.9887539744377136,
0.9417094588279724,
0.9137468934059143,
0.8887874484062195,
0.870714545249939,
0.8622097379673 0042,
0.8365942239761353,
0.8060234785079956,
0.8074963092803955,
0.7745527029037476,
0.7541061639785767,
0.7390430569648743,
0.6817983388900757,
0.6512948274612427,
0.6580785316467285,
0.6259148120880127,
0.5624890923500061,
0.5113409757614136,
0.4763440787792206,
0.4805532097816 4673,
0.4506080059604248,
0.4069921076297 76,
0.3540780544281006,
0.3384069800376892,
0.3504347205162 0483,
0.3083182871134 17053,
0.2923457374226166,
0.2777604758739 4714,
0.3139457404613495,
0.2472945451736 4502,
0.2291684299707 4127,
0.2676651477813 7207,
0.2832731008529663,
0.2170590907335 2814,
0.1878115087747 5739,
0.1453621536493 3014,
0.1256638020277 0233,
0.1233699619770 05,
0.1103141970838 0643,
0.1348487734794 6167,
0.1302433907985 6873,
0.1090921834111 2137,
0.1087516769766 8076,
0.0960841178894043,
0.0744242742657 6614,
0.0750354602932 93,
0.0931398496031 7612,
0.1130263060331 3446,
0.1043308079242 7063,
0.0944361165516 59012,
0.0874404534697 5327,
0.0871712788939476,
0.0993364602327 3468,
0.0861590206230 774,
0.0864851623773 5748,
0.0762897953391 0751,
0.0726614221930 9038,
0.0717584118247 0322,
0.0751463323831 5582,
0.0700325071811 676,
0.0666837245225 9064,
0.0673092305660 2478,
0.0569041997528 0762,
0.0661602169275 2838,
0.0872352719306 9458,
0.0911646261811 2564,
0.0809356570243 8354,
0.0609704703092 5751,
0.0638023316860 199,
0.0642500296235 0845,
0.0720387473702 4307,
0.0694999843833 58307,
0.0838321894407 2723,
0.0996306985616 684,
0.1308675706386 5662,
0.1483175903558 731,
0.1536699682474 1364,
0.2511159090332 031,
0.2518834769725 7996,
0.2440163046121 5973,
0.2699933350806 212,
0.2767392992973 3276,
0.2311156094074 2493,
0.1699163764715 1947,
0.1362795084714 8895,
0.1103795990347 8622,
0.0924827539080 1102,

```
0.08575712889432907,
0.06993220001459122,
0.06178823113441467,
0.04699347913265228,
0.03826254978775978,
0.03089750185608864,
0.027607811614871025,
0.02927281893789768 2,
0.026825210079550743,
0.020687835291020023],
'val_loss': [1.3523014783859253,
1.3695198297500 61,
1.1390112638473 51,
1.4328569173812866,
1.2451062202453613,
1.1284995079044527,
1.1435775756835938,
1.1828629970550537,
1.1236282587051392,
1.0858964920043945,
1.0566132068634033,
1.0468471050262451,
1.070696234703064,
1.2483698129653 93,
1.1707909107208252,
1.2608816623687744,
1.1172431707382202,
1.2004870176315308,
1.2835153341293335,
1.2995040416717 53,
1.2744839191436768,
1.2451573610305786,
1.4742132425308228,
1.2805343866348267,
1.3945769071578 98,
1.4239073991775513,
1.4364521503448486,
1.4626481533050537,
1.6037241220474243,
1.5551298896270752,
1.6725780963897705,
1.4546945095062256,
1.5642159064048767,
1.5199230909347534,
1.3482711315155 03,
1.5039179325103 76,
1.6250413656234741,
1.4969272613525 39,
1.5514297485351562,
1.5651320219039917,
1.5103418827056885,
1.5151708126068115,
1.4903613328933716,
1.5195720195770264,
1.5008760943435 67,
1.5410983562469482,
1.5359649658203125,
1.5291416645050049,
1.5412895679473877,
1.5389355421066284,
1.5876601934432983,
1.5732009410858154,
1.5174937242299 8,
1.5326201915740967,
1.5607825551765442,
1.5859292745590 21,
1.5621286630630493,
1.5689367055892944,
1.5171382427215576,
1.5943368282674316,
1.5699539184570312,
1.5748839378356934,
1.5445791482925415,
1.5586063861846924,
1.5271600484848022,
1.5266829729080 2,
1.5345155000686646,
1.4946649410247802 7,
1.5908048759277344,
1.5287828444543457,
1.5119296312332153,
1.4940084218978882,
1.5362843275070 19,
1.5374985933303833,
1.5240010023117065,
1.5013190507888794,
1.5502082109451294,
1.5146405696868896,
1.5284087657928467,
1.5161666870117188,
1.5771233442459106,
1.4031710624694824,
1.4747451543807983,
1.4158124788284302,
1.6154534816741943,
1.5397838354110718,
1.4532064199447632,
1.4613121747970058,
1.5166007280349731,
1.5206322669982910,
1.5245395898881897,
1.5278993844985962,
1.4940354824066162,
1.5090551376342773,
1.5007407665252686,
1.4958673580169678,
1.4798171520233154,
1.4940725564956665,
1.4962961673736572,
1.4879598617553711],
'val_root_mean_squared_error': [1.1628849506378174,
1.1702648401260376,
1.0672446489334106,
1.1970200538635254,
1.1158432960510254,
1.0623085498809814,
1.0693818330076477,
1.0875951051712036,
1.0600132942199707,
1.0420635938644441,
1.0279169082641602,
1.0231554508209229,
1.0347445011138916,
1.1173046827316284,
1.0820308923721313,
1.1228898763656616,
1.0569972991943 36,
1.0956673622131348,
1.1329233646392822,
1.1399579048156738,
1.1289304494857788,
1.1158661842346191,
1.2141718864440918,
1.1316069336547 73,
1.1809220314025 88,
1.1932759284973145,
1.1985207796096802,
1.2093999385833 74,
1.2663823366165161,
1.2470482587814 33,
1.2932819128036 5,
1.2061071395874023,
1.2506858110427856,
1.2328516244888306,
1.1611508131027222,
1.2263432741165161,
1.2747710943222046,
1.2234897613525 39,
1.2455639839172363,
1.2510523796081543,
1.2289596796035767,
1.2309226989746094,
1.2208036184310913,
1.2327091693878174,
1.2280385494232178,
1.2414097785949707,
1.2393405437469482,
1.2365846633911133,
1.2414867877960205,
1.2405835868835545,
1.2600238323211670,
1.2542730569839478,
1.2318660020828247,
1.2379903793334 96,
1.2493128776550293,
1.2593368291854858,
1.2498514652252197,
1.2525720596313477,
1.2317216396331787,
1.2626701593399048,
1.2529779672622 68,
1.2549438476562 5,
1.2428109645843506,
1.2484415769577026,
1.2357832193374634,
1.2355901002883911,
1.2387555837631226,
1.2225584983825684,
1.2609853744506836,
1.2364395856857 3,
1.2296054363250732,
1.2222963571548462,
1.2394694089889526,
1.2399591207504272,
1.2345043420791626,
1.2252832651138306,
1.2458735569000244,
1.2307072877883911,
1.2362883090972 9,
1.2313271760940552,
```

```
1.2558393478393555,
1.1845551729920227,
1.2143908739089966,
1.1895430088043213,
1.2710049152374268,
1.2408802509307861,
1.2054901123046875,
1.2088474035263062,
1.2315034866333008,
1.2331391572952227,
1.2347224950790405,
1.2360823154449463,
1.2223074436187744,
1.2284361124038696,
1.225047230720252,
1.2227294445037842,
1.216477394104004,
1.2223225831985474,
1.2232319116592407,
1.2198195457458496]}
```

In [81]:
```
# Посмотрим на график потерь на тренировочной и тестовой выборках
def model_loss_plot(model_hist1):
    plt.figure(figsize = (17,5))
    plt.plot(model_hist1.history['loss'],
             label = 'ошибка на обучающей выборке')
    plt.plot(model_hist1.history['val_loss'],
             label = 'ошибка на тестовой выборке')
    plt.title('График потерь модели')
    plt.ylabel('Значение ошибки')
    plt.xlabel('Эпохи')
    plt.legend(['Ошибка на обучающей выборке', 'Ошибка на тестовой выборке'], loc='best')
    plt.show()
model_loss_plot(model_hist1)
```



In [78]:
```
# Зададим функцию для визуализации факт/прогноз для результатов моделей
# Посмотрим на график результата работы модели
def actual_and_predicted_plot(orig, predict, var, model_name):
    plt.figure(figsize=(17,5))
    plt.title(f'Тестовые и прогнозные значения: {model_name}')
    plt.plot(orig, label = 'Тест')
    plt.plot(predict, label = 'Прогноз')
    plt.legend(loc = 'best')
    plt.ylabel(var)
    plt.xlabel('Количество наблюдений')
    plt.show()
actual_and_predicted_plot(y_test.values, model1.predict(x_test.values), 'Соотношение матрица/наполнитель', 'Keras_neuronet')
```

```
9/9 [==============================] - 0s 2ms/step
```


In [79]:
```
# оценка модели MSE
model1.evaluate(x_test, y_test, verbose = 1)
```

```
9/9 [==============================] - 0s 2ms/step - loss: 1.2427 - root_mean_squared_error: 1.1147
```

Out[79]:
```
[1.2426555156707764, 1.1147445440292358]
```

In [80]:
```
test_predictions = model1.predict(x_test).flatten()

a = plt.axes(aspect = 'equal')
plt.scatter(y_test, test_predictions)
plt.xlabel('True Values')
plt.ylabel('Predictions')
lims = [0, 5]
plt.xlim(lims)
plt.ylim(lims)
_ = plt.plot(lims, lims)
```

```
9/9 [==============================] - 0s 2ms/step
```


Заключение.

Подводя итоги, стоит сказать, что машинное обучение в задачах моделей прогнозирования – довольно сложный процесс, требующий не только навыков программирования, но и профессионального подхода к сфере самих композитных материалов. Необходимо понимать, на какие атрибуты нужно в первую очередь обратить внимание, чтобы суметь впоследствии грамотно и чётко спрогнозировать тот или иной признак. И, естественно, обладать всеми необходимыми знаниями, умениями и навыками для прогнозов и расчётов. В ходе работы был задействован дата-сет с реальными данными, произведена его подробная опись и сопутствующий анализ; построено множество разнообразных графиков; осуществлено разбиение данных на обучающую и тестовую выборки с использованием множества вспомогательных модулей из библиотеки SkLearn, которая во многом облегчила процесс машинного обучения и в целом была очень полезным инструментом в ходе работы над выпускной квалификационной работой. В рамках машинного обучения и поиска гиперпараметров были задействованы несколько алгоритмов: линейная регрессия, градиентный бустинг, K ближайших соседей, деревья решений, стохастический градиентный спуск, многослойный перцептрон, лассо регрессия, а также опорные вектора и случайный лес. Поиск гиперпараметров осуществлялся при помощи таких методов, как «GridSearch». Для каждой из выборок были составлены классификационные отчёты, содержащие в себе основополагающие метрики, оценивающие качество проводимого обучения. В конечном итоге было представлено сравнение результатов оценок работы алгоритмов, а также различные графики и диаграммы, позволяющие наглядно оценить итоги проведенного обучения. Обучена нейронная сеть и разработано пользовательское приложение, предсказывающее вероятный прогноз по заданным параметрам. Что касается перспектив решения данной проблемы композитных материалов, то я думаю, что в таких случаях необходимо уделить больше внимания изучению самой проблемы композитных материалов, углубить знания по статистике и регрессиям, поискать иные варианты решений с данным датасетом, создать плодтворную команду программистов и сотрудников, работающих с природными материалами, способную к совместной работе над усовершенствованием уже существующих разработок и поддержанием их качественного и бесперебойного функционирования.

In [ ]:
```
```