Salaryregression

April 7, 2025

0.1 ANN Regression Practical Implementation

[1]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder, OneHotEncoder
import pickle
```

[2]:
```python
data = pd.read_csv('../Datasets/Churn_Modelling.csv')
data.head()
```

[2]:
```
   RowNumber  CustomerId   Surname  CreditScore Geography  Gender  Age  \
0          1    15634602  Hargrave          619    France  Female   42
1          2    15647311      Hill          608     Spain  Female   41
2          3    15619304      Onio          502    France  Female   42
3          4    15701354      Boni          699    France  Female   39
4          5    15737888  Mitchell          850     Spain  Female   43

   Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0       2       0.00              1          1               1
1       1   83807.86              1          0               1
2       8  159660.80              3          1               0
3       1       0.00              2          0               0
4       2  125510.82              1          1               1

   EstimatedSalary  Exited
0        101348.88       1
1        112542.58       0
2        113931.57       1
3         93826.63       0
4         79084.10       0
```

[3]:
```python
## Preprocess the data
data = data.drop(['RowNumber', 'CustomerId', 'Surname'], axis=1)
```

[4]:
```python
## Define the label encoder
label_encoder_gender = LabelEncoder()
data['Gender'] = label_encoder_gender.fit_transform(data['Gender'])
```

[5]:
```python
# One-hot encoder for the geography
onehot_encoder_geo = OneHotEncoder(handle_unknown='ignore')
geo_encoded = onehot_encoder_geo.fit_transform(data[['Geography']]).toarray()
geo_encoded_df = pd.DataFrame(geo_encoded, columns=onehot_encoder_geo.
 ↪get_feature_names_out(['Geography']))
geo_encoded_df
```

[5]:
```
      Geography_France  Geography_Germany  Geography_Spain
0                  1.0                0.0              0.0
1                  0.0                0.0              1.0
2                  1.0                0.0              0.0
3                  1.0                0.0              0.0
4                  0.0                0.0              1.0
…                    …                  …                …
9995               1.0                0.0              0.0
9996               1.0                0.0              0.0
9997               1.0                0.0              0.0
9998               0.0                1.0              0.0
9999               1.0                0.0              0.0

[10000 rows x 3 columns]
```

[6]:
```python
## Combining the one hot encoded geography to original data
data = pd.concat([data.drop('Geography', axis=1), geo_encoded_df], axis=1)
data
```

[6]:
```
      CreditScore  Gender  Age  Tenure    Balance  NumOfProducts  HasCrCard  \
0             619       0   42       2       0.00              1          1
1             608       0   41       1   83807.86              1          0
2             502       0   42       8  159660.80              3          1
3             699       0   39       1       0.00              2          0
4             850       0   43       2  125510.82              1          1
…               …     …   …       …          …              …
9995          771       1   39       5       0.00              2          1
9996          516       1   35      10   57369.61              1          1
9997          709       0   36       7       0.00              1          0
9998          772       1   42       3   75075.31              2          1
9999          792       0   28       4  130142.79              1          1

      IsActiveMember  EstimatedSalary  Exited  Geography_France  \
0                  1        101348.88       1               1.0
1                  1        112542.58       0               0.0
2                  0        113931.57       1               1.0
3                  0         93826.63       0               1.0
4                  1         79084.10       0               0.0
…                  …                …     …                 …
9995               0         96270.64       0               1.0
```

```
9996                 1        101699.77       0                1.0
9997                 1         42085.58       1                1.0
9998                 0         92888.52       1                0.0
9999                 0         38190.78       0                1.0

      Geography_Germany  Geography_Spain
0                   0.0              0.0
1                   0.0              1.0
2                   0.0              0.0
3                   0.0              0.0
4                   0.0              1.0
…                  …               …
9995                0.0              0.0
9996                0.0              0.0
9997                0.0              0.0
9998                1.0              0.0
9999                0.0              0.0

[10000 rows x 13 columns]
```

[7]:
```
# Split the data in features and target
X = data.drop('EstimatedSalary', axis=1)
y = data['EstimatedSalary']
```

[8]:
```
# Split the data in training and testing
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[9]:
```
# Standard Scaler
from sklearn.preprocessing import StandardScaler, LabelEncoder
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[10]:
```
#  Save the Encoders and scaler for user
with open('label_encoder_gender.pkl', 'wb') as file:
        pickle.dump(label_encoder_gender, file)

with open('onehot_encoder_geo.pkl', 'wb') as file:
        pickle.dump(onehot_encoder_geo, file)

with open('scaler.pkl', 'wb') as file:
        pickle.dump(scaler, file)
```

0.1.1 ANN Regression Problem Statement

[11]:
```
X_train.shape[1],
```

[11]: (12,)

[12]:
```
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (8000, 12)
y_train shape: (8000,)
X_test shape: (2000, 12)
y_test shape: (2000,)
```

[13]:
```
import tensorflow as tf
print(tf.__version__)
```

```
2.19.0
```

[14]:
```
import tensorflow as tf
from tensorflow.keras.callbacks import EarlyStopping, TensorBoard
import datetime
# Setup of tensorboard
log_dir="regressionlogs/fit" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback=TensorBoard(log_dir=log_dir, histogram_freq=1)
```

[15]:
```
## setup Early Stopping
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=10,␣
↪restore_best_weights=True)
```

[16]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import numpy as np

# Step 2: Ensure data is in correct format
X_train = np.array(X_train, dtype=np.float32)
X_test = np.array(X_test, dtype=np.float32)
y_train = np.array(y_train, dtype=np.float32).reshape(-1, 1)
y_test = np.array(y_test, dtype=np.float32).reshape(-1, 1)

# Step 3: Clear any previous TensorFlow models
tf.keras.backend.clear_session()

# Step 4: Build the model with correct layer names
model = Sequential([
```

```python
    Dense(64, activation='relu', input_shape=(X_train.shape[1],),␣
 ↪name='InputLayer'),
    Dense(32, activation='relu', name='HiddenLayer1'),
    Dense(1, name='OutputLayer')
])

model.summary()

# Step 5: Compile the model
model.compile(optimizer='adam', loss='mean_absolute_error', metrics=['mae'])

# Step 6: Train the model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test),␣
 ↪epochs=100, callbacks=[early_stopping_callback, tensorboard_callback])
```

```
WARNING:tensorflow:From c:\Users\Uditya\Desktop\Deep Learning\Deep Learning for
Beginner\venv\lib\site-packages\keras\src\backend\common\global_state.py:82: The
name tf.reset_default_graph is deprecated. Please use
tf.compat.v1.reset_default_graph instead.


c:\Users\Uditya\Desktop\Deep Learning\Deep Learning for Beginner\venv\lib\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
InputLayer (Dense)	(None, 64)	832
HiddenLayer1 (Dense)	(None, 32)	2,080
OutputLayer (Dense)	(None, 1)	33

Total params: 2,945 (11.50 KB)

Trainable params: 2,945 (11.50 KB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/100
```

```
250/250             6s 11ms/step -
loss: 100591.4062 - mae: 100591.4062 - val_loss: 98584.9141 - val_mae:
98584.9141
Epoch 2/100
250/250             4s 8ms/step -
loss: 99912.1016 - mae: 99912.1016 - val_loss: 97516.2109 - val_mae: 97516.2109
Epoch 3/100
250/250             3s 10ms/step -
loss: 98909.4766 - mae: 98909.4766 - val_loss: 94650.1797 - val_mae: 94650.1797
Epoch 4/100
250/250             3s 10ms/step -
loss: 95055.3203 - mae: 95055.3203 - val_loss: 89670.6172 - val_mae: 89670.6172
Epoch 5/100
250/250             3s 11ms/step -
loss: 89792.1484 - mae: 89792.1484 - val_loss: 82875.3984 - val_mae: 82875.3984
Epoch 6/100
250/250             3s 10ms/step -
loss: 81031.4141 - mae: 81031.4141 - val_loss: 74977.4062 - val_mae: 74977.4062
Epoch 7/100
250/250             3s 11ms/step -
loss: 74227.1250 - mae: 74227.1250 - val_loss: 67322.1406 - val_mae: 67322.1406
Epoch 8/100
250/250             4s 8ms/step -
loss: 65635.4531 - mae: 65635.4531 - val_loss: 60786.2969 - val_mae: 60786.2969
Epoch 9/100
250/250             3s 12ms/step -
loss: 60026.8086 - mae: 60026.8086 - val_loss: 56009.2891 - val_mae: 56009.2891
Epoch 10/100
250/250             4s 9ms/step -
loss: 55452.3984 - mae: 55452.3984 - val_loss: 53124.8711 - val_mae: 53124.8711
Epoch 11/100
250/250             2s 7ms/step -
loss: 53442.6055 - mae: 53442.6055 - val_loss: 51669.7539 - val_mae: 51669.7539
Epoch 12/100
250/250             1s 6ms/step -
loss: 51773.5469 - mae: 51773.5469 - val_loss: 51077.4922 - val_mae: 51077.4922
Epoch 13/100
250/250             2s 6ms/step -
loss: 50434.9883 - mae: 50434.9883 - val_loss: 50839.7695 - val_mae: 50839.7695
Epoch 14/100
250/250             2s 9ms/step -
loss: 50596.1523 - mae: 50596.1523 - val_loss: 50728.9805 - val_mae: 50728.9805
Epoch 15/100
250/250             2s 7ms/step -
loss: 50329.8398 - mae: 50329.8398 - val_loss: 50665.2852 - val_mae: 50665.2852
Epoch 16/100
250/250             2s 8ms/step -
loss: 50020.5664 - mae: 50020.5664 - val_loss: 50617.6797 - val_mae: 50617.6797
```

```
Epoch 17/100
250/250            2s 9ms/step -
loss: 49703.2500 - mae: 49703.2500 - val_loss: 50579.2227 - val_mae: 50579.2227
Epoch 18/100
250/250            2s 6ms/step -
loss: 50002.0898 - mae: 50002.0898 - val_loss: 50551.1602 - val_mae: 50551.1602
Epoch 19/100
250/250            2s 7ms/step -
loss: 49847.8906 - mae: 49847.8906 - val_loss: 50526.5312 - val_mae: 50526.5312
Epoch 20/100
250/250            2s 7ms/step -
loss: 50195.5938 - mae: 50195.5938 - val_loss: 50502.4766 - val_mae: 50502.4766
Epoch 21/100
250/250            2s 9ms/step -
loss: 50186.5391 - mae: 50186.5391 - val_loss: 50483.1250 - val_mae: 50483.1250
Epoch 22/100
250/250            2s 8ms/step -
loss: 49615.1758 - mae: 49615.1758 - val_loss: 50461.4375 - val_mae: 50461.4375
Epoch 23/100
250/250            2s 8ms/step -
loss: 49833.5156 - mae: 49833.5156 - val_loss: 50444.1016 - val_mae: 50444.1016
Epoch 24/100
250/250            2s 9ms/step -
loss: 49531.4141 - mae: 49531.4141 - val_loss: 50421.2773 - val_mae: 50421.2773
Epoch 25/100
250/250            3s 11ms/step -
loss: 49690.7617 - mae: 49690.7617 - val_loss: 50413.7461 - val_mae: 50413.7461
Epoch 26/100
250/250            2s 8ms/step -
loss: 49479.6016 - mae: 49479.6016 - val_loss: 50396.5273 - val_mae: 50396.5273
Epoch 27/100
250/250            4s 12ms/step -
loss: 49547.0508 - mae: 49547.0508 - val_loss: 50384.2852 - val_mae: 50384.2852
Epoch 28/100
250/250            2s 7ms/step -
loss: 49538.8320 - mae: 49538.8320 - val_loss: 50372.6758 - val_mae: 50372.6758
Epoch 29/100
250/250            2s 10ms/step -
loss: 49627.8164 - mae: 49627.8164 - val_loss: 50363.9922 - val_mae: 50363.9922
Epoch 30/100
250/250            2s 8ms/step -
loss: 49673.6055 - mae: 49673.6055 - val_loss: 50361.3984 - val_mae: 50361.3984
Epoch 31/100
250/250            2s 8ms/step -
loss: 49897.9727 - mae: 49897.9727 - val_loss: 50360.9141 - val_mae: 50360.9141
Epoch 32/100
250/250            2s 10ms/step -
loss: 49558.7539 - mae: 49558.7539 - val_loss: 50349.1641 - val_mae: 50349.1641
```

```
Epoch 33/100
250/250            2s 7ms/step -
loss: 50180.0312 - mae: 50180.0312 - val_loss: 50348.1758 - val_mae: 50348.1758
Epoch 34/100
250/250            2s 8ms/step -
loss: 49953.9414 - mae: 49953.9414 - val_loss: 50338.5977 - val_mae: 50338.5977
Epoch 35/100
250/250            2s 8ms/step -
loss: 50039.2969 - mae: 50039.2969 - val_loss: 50341.1719 - val_mae: 50341.1719
Epoch 36/100
250/250            3s 10ms/step -
loss: 49701.7500 - mae: 49701.7500 - val_loss: 50340.4141 - val_mae: 50340.4141
Epoch 37/100
250/250            3s 11ms/step -
loss: 49594.4727 - mae: 49594.4727 - val_loss: 50337.8164 - val_mae: 50337.8164
Epoch 38/100
250/250            6s 13ms/step -
loss: 49745.1094 - mae: 49745.1094 - val_loss: 50332.1523 - val_mae: 50332.1523
Epoch 39/100
250/250            2s 10ms/step -
loss: 49355.3828 - mae: 49355.3828 - val_loss: 50325.9219 - val_mae: 50325.9219
Epoch 40/100
250/250            3s 11ms/step -
loss: 49767.6367 - mae: 49767.6367 - val_loss: 50327.2383 - val_mae: 50327.2383
Epoch 41/100
250/250            2s 7ms/step -
loss: 49710.1055 - mae: 49710.1055 - val_loss: 50321.1328 - val_mae: 50321.1328
Epoch 42/100
250/250            4s 13ms/step -
loss: 49398.2773 - mae: 49398.2773 - val_loss: 50319.8906 - val_mae: 50319.8906
Epoch 43/100
250/250            4s 7ms/step -
loss: 49218.0781 - mae: 49218.0781 - val_loss: 50322.4023 - val_mae: 50322.4023
Epoch 44/100
250/250            2s 8ms/step -
loss: 49832.0039 - mae: 49832.0039 - val_loss: 50319.6523 - val_mae: 50319.6523
Epoch 45/100
250/250            3s 8ms/step -
loss: 49873.3516 - mae: 49873.3516 - val_loss: 50316.3008 - val_mae: 50316.3008
Epoch 46/100
250/250            2s 10ms/step -
loss: 49523.1680 - mae: 49523.1680 - val_loss: 50315.5234 - val_mae: 50315.5234
Epoch 47/100
250/250            2s 9ms/step -
loss: 49213.9102 - mae: 49213.9102 - val_loss: 50316.6523 - val_mae: 50316.6523
Epoch 48/100
250/250            2s 8ms/step -
loss: 49632.1406 - mae: 49632.1406 - val_loss: 50317.2969 - val_mae: 50317.2969
```

```
Epoch 49/100
250/250              3s 10ms/step -
loss: 49851.3125 - mae: 49851.3125 - val_loss: 50316.1914 - val_mae: 50316.1914
Epoch 50/100
250/250              2s 7ms/step -
loss: 49568.7734 - mae: 49568.7734 - val_loss: 50311.0391 - val_mae: 50311.0391
Epoch 51/100
250/250              2s 8ms/step -
loss: 49461.8555 - mae: 49461.8555 - val_loss: 50313.0391 - val_mae: 50313.0391
Epoch 52/100
250/250              2s 7ms/step -
loss: 49598.8438 - mae: 49598.8438 - val_loss: 50305.5156 - val_mae: 50305.5156
Epoch 53/100
250/250              2s 8ms/step -
loss: 49547.0508 - mae: 49547.0508 - val_loss: 50302.7930 - val_mae: 50302.7930
Epoch 54/100
250/250              3s 10ms/step -
loss: 49215.2383 - mae: 49215.2383 - val_loss: 50299.7461 - val_mae: 50299.7461
Epoch 55/100
250/250              2s 7ms/step -
loss: 49741.6797 - mae: 49741.6797 - val_loss: 50308.3555 - val_mae: 50308.3555
Epoch 56/100
250/250              3s 8ms/step -
loss: 49395.3438 - mae: 49395.3438 - val_loss: 50318.0781 - val_mae: 50318.0781
Epoch 57/100
250/250              2s 7ms/step -
loss: 49275.7656 - mae: 49275.7656 - val_loss: 50310.2734 - val_mae: 50310.2734
Epoch 58/100
250/250              2s 9ms/step -
loss: 48856.8477 - mae: 48856.8477 - val_loss: 50317.0586 - val_mae: 50317.0586
Epoch 59/100
250/250              2s 9ms/step -
loss: 49401.4414 - mae: 49401.4414 - val_loss: 50320.7656 - val_mae: 50320.7656
Epoch 60/100
250/250              2s 8ms/step -
loss: 49487.8594 - mae: 49487.8594 - val_loss: 50313.5391 - val_mae: 50313.5391
Epoch 61/100
250/250              2s 8ms/step -
loss: 49391.2227 - mae: 49391.2227 - val_loss: 50318.3945 - val_mae: 50318.3945
Epoch 62/100
250/250              2s 8ms/step -
loss: 49052.2500 - mae: 49052.2500 - val_loss: 50309.7695 - val_mae: 50309.7695
Epoch 63/100
250/250              3s 10ms/step -
loss: 49893.3438 - mae: 49893.3438 - val_loss: 50317.3633 - val_mae: 50317.3633
Epoch 64/100
250/250              5s 11ms/step -
loss: 49339.8984 - mae: 49339.8984 - val_loss: 50311.2188 - val_mae: 50311.2188
```

[17]:
```
%load_ext tensorboard
```

[25]:
```
%tensorboard --logdir regressionlogs/fit20250406-204615
```

```
Reusing TensorBoard on port 6007 (pid 5060), started 0:00:07 ago. (Use '!kill␣
↪5060' to kill it.)
```

```
<IPython.core.display.HTML object>
```

[19]:
```
## Evaluate the model on test data
test_loss, test_mae = model.evaluate(X_test, y_test)
print(f'Test_MAE {test_mae}')
```

```
63/63              0s 4ms/step - loss:
51113.4570 - mae: 51113.4570
Test_MAE 50299.74609375
```

[20]:
```
model.save('regression.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
```

0.1.2 By increasing of layers the model learns better

[21]:
```
from tensorflow.keras.layers import LeakyReLU

model = Sequential([
    Dense(128, input_shape=(X_train.shape[1],), name='InputLayer'),
    LeakyReLU(alpha=0.1),

    Dense(64, name='FirstHiddenLayer'),
    LeakyReLU(alpha=0.1),

    Dense(32,name='SecondHiddenLayer'),
    LeakyReLU(alpha=0.1),

    Dense(1, name='OutputLayer')
])
```

```
c:\Users\Uditya\Desktop\Deep Learning\Deep Learning for Beginner\venv\lib\site-
packages\keras\src\layers\activations\leaky_relu.py:41: UserWarning: Argument
`alpha` is deprecated. Use `negative_slope` instead.
  warnings.warn(
```

[22]:
```
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
InputLayer (Dense)	(None, 128)	1,664
leaky_re_lu (LeakyReLU)	(None, 128)	0
FirstHiddenLayer (Dense)	(None, 64)	8,256
leaky_re_lu_1 (LeakyReLU)	(None, 64)	0
SecondHiddenLayer (Dense)	(None, 32)	2,080
leaky_re_lu_2 (LeakyReLU)	(None, 32)	0
OutputLayer (Dense)	(None, 1)	33

Total params: 12,033 (47.00 KB)

Trainable params: 12,033 (47.00 KB)

Non-trainable params: 0 (0.00 B)

[23]:
```
from tensorflow.keras.callbacks import EarlyStopping

early_stop = EarlyStopping(monitor='val_loss', patience=10,␣
↪restore_best_weights=True)

model.compile(optimizer='adam', loss='mean_absolute_error', metrics=['mae'])

history = model.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=100,
    batch_size=32,
    callbacks=[early_stop]
)
```

```
Epoch 1/100
250/250             6s 7ms/step -
loss: 99269.6797 - mae: 99269.6797 - val_loss: 83595.3750 - val_mae: 83595.3750
Epoch 2/100
```

```
250/250              2s 7ms/step -
loss: 71407.0469 - mae: 71407.0469 - val_loss: 50281.8477 - val_mae: 50281.8477
Epoch 3/100
250/250              3s 8ms/step -
loss: 49550.1953 - mae: 49550.1953 - val_loss: 50326.9570 - val_mae: 50326.9570
Epoch 4/100
250/250              2s 7ms/step -
loss: 50154.7031 - mae: 50154.7031 - val_loss: 50269.2188 - val_mae: 50269.2188
Epoch 5/100
250/250              2s 9ms/step -
loss: 49554.1250 - mae: 49554.1250 - val_loss: 50260.6133 - val_mae: 50260.6133
Epoch 6/100
250/250              2s 7ms/step -
loss: 49607.3203 - mae: 49607.3203 - val_loss: 50212.0781 - val_mae: 50212.0781
Epoch 7/100
250/250              2s 6ms/step -
loss: 49838.6875 - mae: 49838.6875 - val_loss: 50270.5312 - val_mae: 50270.5312
Epoch 8/100
250/250              2s 7ms/step -
loss: 49863.3047 - mae: 49863.3047 - val_loss: 50194.8711 - val_mae: 50194.8711
Epoch 9/100
250/250              3s 11ms/step -
loss: 49177.2656 - mae: 49177.2656 - val_loss: 50220.0078 - val_mae: 50220.0078
Epoch 10/100
250/250              2s 7ms/step -
loss: 49569.1172 - mae: 49569.1172 - val_loss: 50222.9688 - val_mae: 50222.9688
Epoch 11/100
250/250              1s 5ms/step -
loss: 49659.4844 - mae: 49659.4844 - val_loss: 50199.7148 - val_mae: 50199.7148
Epoch 12/100
250/250              1s 5ms/step -
loss: 49326.4609 - mae: 49326.4609 - val_loss: 50276.0898 - val_mae: 50276.0898
Epoch 13/100
250/250              2s 6ms/step -
loss: 48771.0664 - mae: 48771.0664 - val_loss: 50231.4961 - val_mae: 50231.4961
Epoch 14/100
250/250              1s 5ms/step -
loss: 49075.1328 - mae: 49075.1328 - val_loss: 50304.6328 - val_mae: 50304.6328
Epoch 15/100
250/250              4s 11ms/step -
loss: 49560.9023 - mae: 49560.9023 - val_loss: 50281.3047 - val_mae: 50281.3047
Epoch 16/100
250/250              2s 6ms/step -
loss: 49179.8633 - mae: 49179.8633 - val_loss: 50285.9062 - val_mae: 50285.9062
Epoch 17/100
250/250              2s 9ms/step -
loss: 49490.5742 - mae: 49490.5742 - val_loss: 50293.2852 - val_mae: 50293.2852
Epoch 18/100
```

```
250/250              2s 7ms/step -
loss: 49241.4609 - mae: 49241.4609 - val_loss: 50232.7773 - val_mae: 50232.7773
```