Copyright Notice

```
model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    tf.keras.layers.Conv1D(filters=32, kernel_size=5,
                           strides=1, padding="causal",
                           activation="relu")
    tf.keras.layers.LSTM(32, return_sequences=True),
    tf.keras.layers.LSTM(32),
    tf.keras.layers.Dense(1),
    tf.keras.layers.Lambda(lambda x: x * 200)
])


optimizer = tf.keras.optimizers.SGD(learning_rate=1e-5, momentum=0.9)

model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])

history = model.fit(dataset, epochs=500)
```

https://www.coursera.org/learn/convolutional-neural-networks-tensorflow

deeplearning.ai TensorFlow in Practice Specialization Go To Course

About Syllabus Reviews Instructors Enrollment Options FAQ

Syllabus - What you will learn from this course

WEEK 1

4 hours to complete

Exploring a Larger Dataset

In the first course in this specialization, you had an introduction to TensorFlow, and how, with its high level APIs you could do basic image classification, an you learned a little bit about Convolutional Neural Networks (ConvNets). In this course you'll go deeper into using ConvNets will real-world data, and learn about techniques that you can u... SHOW ALL

8 videos (Total 18 min), 5 readings, 3 quizzes SEE ALL

WEEK 2

4 hours to complete

Augmentation: A technique to avoid overfitting

You've heard the term overfitting a number of times to this point. Overfitting is simply the concept of being over specialized in training -- namely that your model is very good at classifying what it is trained for, but not so good at classifying things that it hasn't seen. In order to generalize your model more effectively, you will of course need a ... SHOW ALL

7 videos (Total 14 min), 6 readings, 3 quizzes SEE ALL

WEEK 3

4 hours to complete

Transfer Learning

Building models for yourself is great, and can be very powerful. But, as you've seen, you can be limited by the data you have on hand. Not everybody has access to massive datasets or the compute power that's needed to train them effectively. Transfer learning can help solve this -- where people with models trained on large datasets train them, ... SHOW ALL

7 videos (Total 14 min), 5 readings, 3 quizzes SEE ALL

WEEK 4

4 hours to complete

Multiclass Classifications

Introduction, A conversation with Andrew Ng

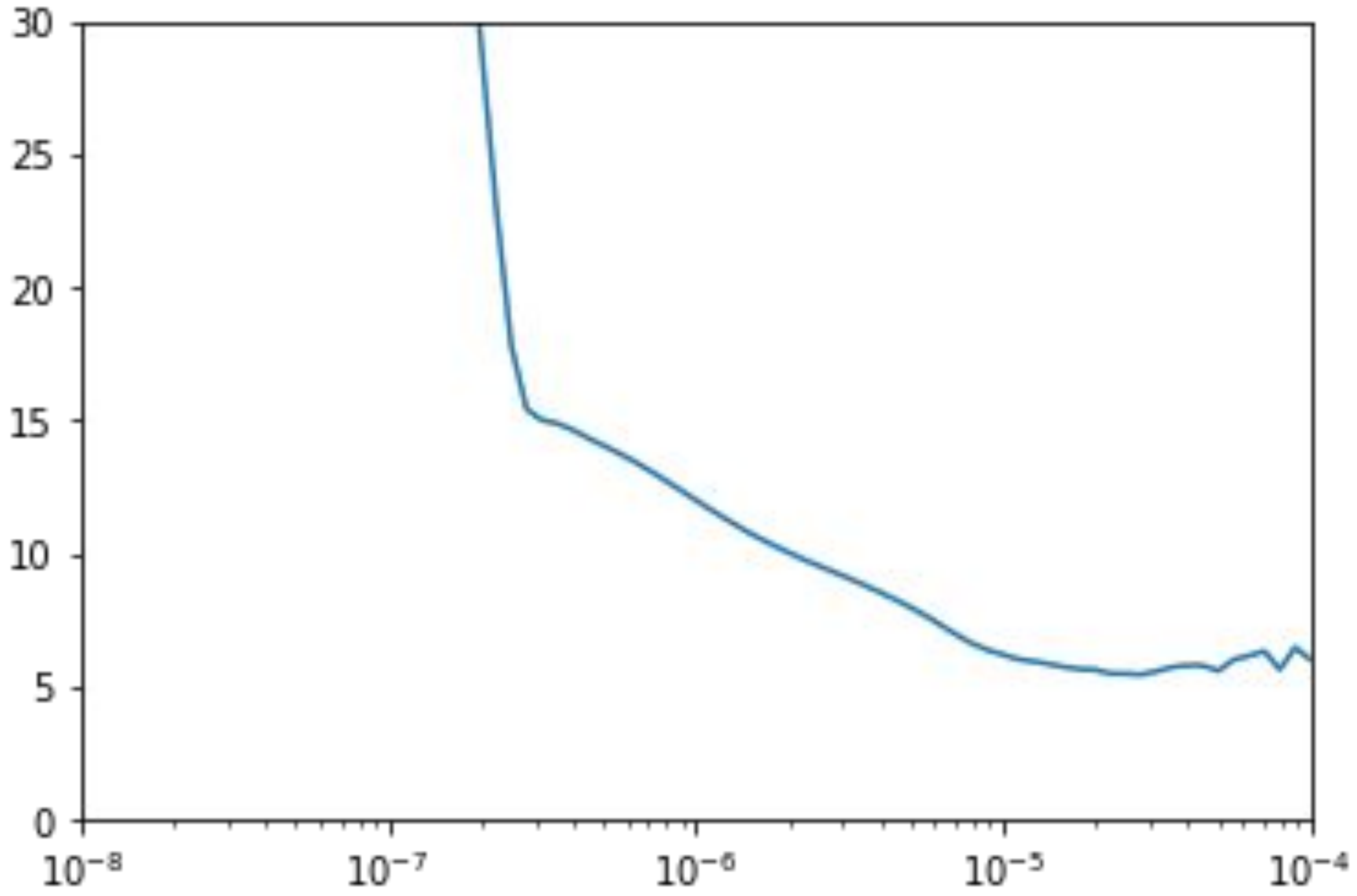
Save Note Discuss Download

English Help Us Translate

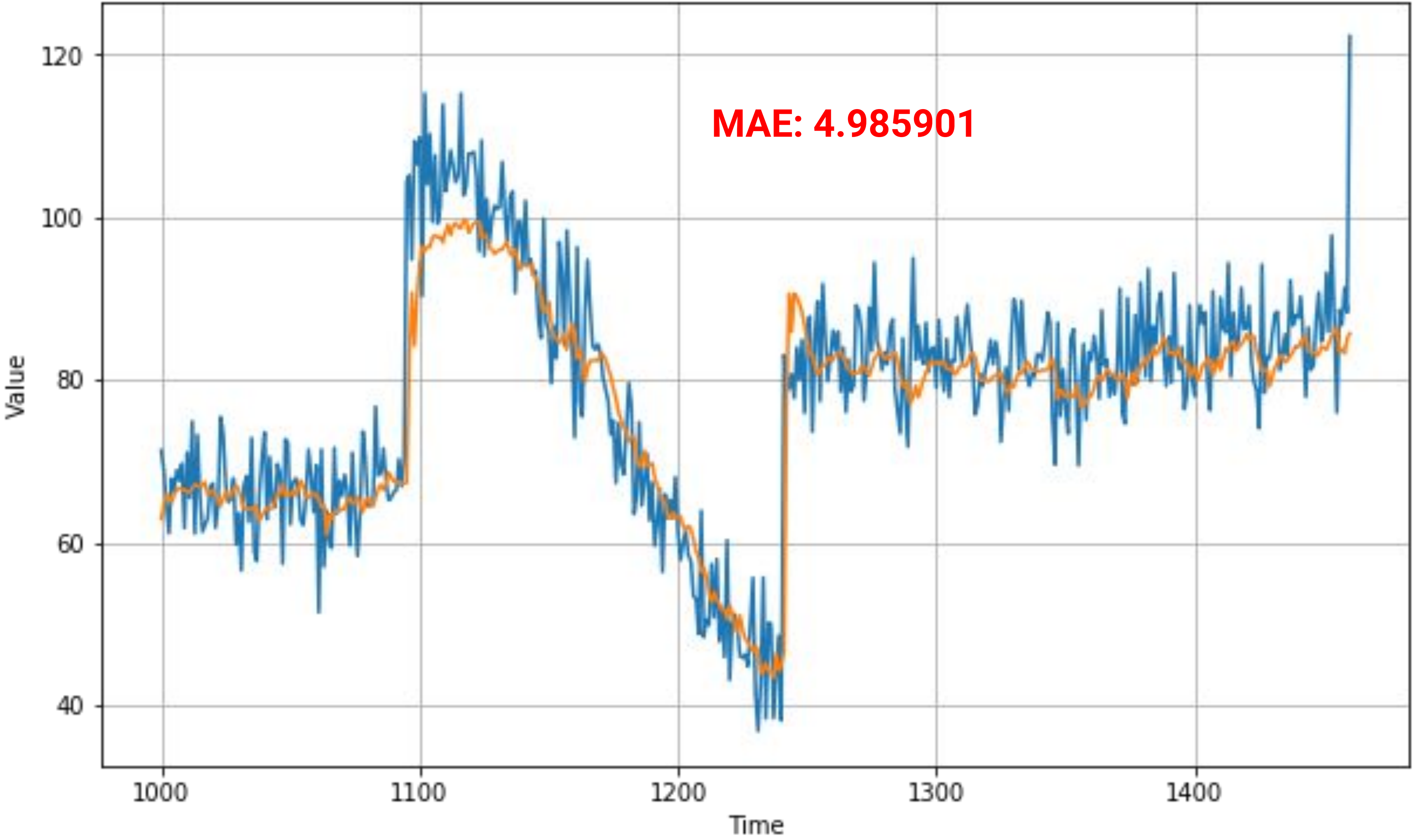
0:00 In the first course, you learned how to use TensorFlow to implement a basic neural network, going up all the way to basic Convolutional Neural Network. In this second course, you go much further. In the first week, you take the ideas you've learned, and apply them to a much bigger dataset of cats versus dogs on Kaggle. Yes so we take the full Kaggle dataset of 25,000 cats versus dogs

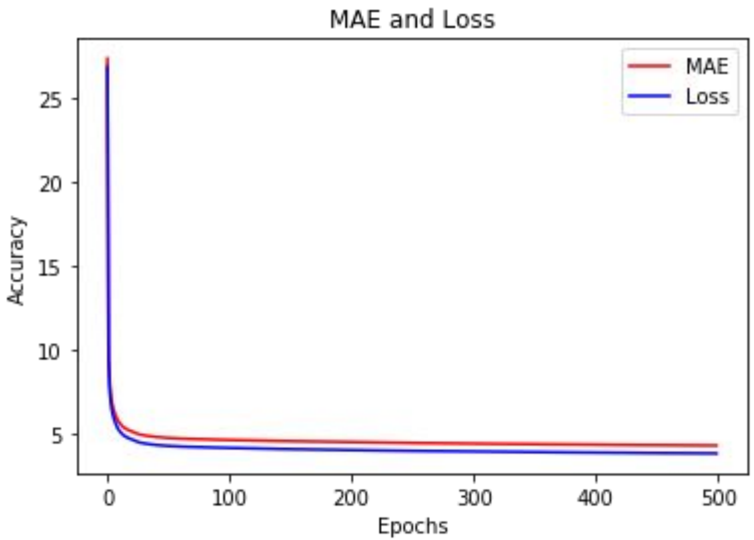
```python
model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    tf.keras.layers.Conv1D(filters=32, kernel_size=5,
                           strides=1, padding="causal",
                           activation="relu")
    tf.keras.layers.LSTM(32, return_sequences=True),
    tf.keras.layers.LSTM(32),
    tf.keras.layers.Dense(1),
    tf.keras.layers.Lambda(lambda x: x * 200)
])

lr_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-8 * 10**(epoch / 20))

optimizer = tf.keras.optimizers.SGD(learning_rate=1e-8, momentum=0.9)

model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])

history = model.fit(dataset, epochs=100, callbacks=[lr_schedule])
```

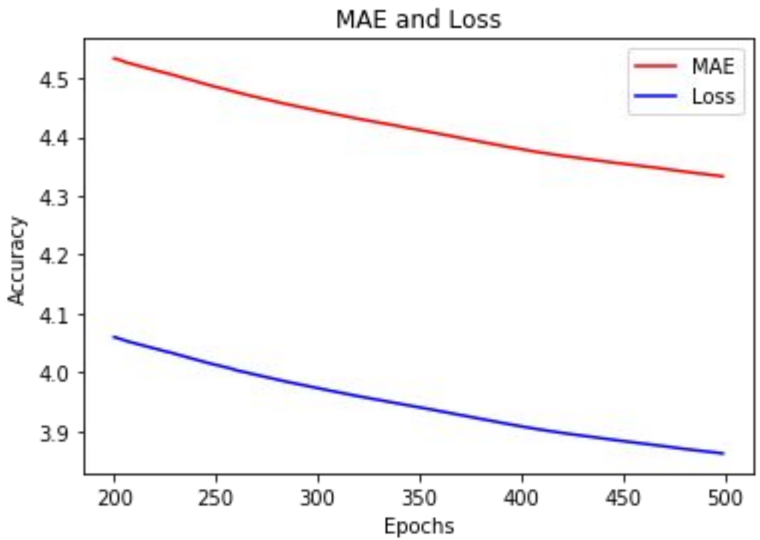
```python
model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    tf.keras.layers.Conv1D(filters=32, kernel_size=5,
                           strides=1, padding="causal",
                           activation="relu")
    tf.keras.layers.LSTM(32, return_sequences=True),
    tf.keras.layers.LSTM(32),
    tf.keras.layers.Dense(1),
    tf.keras.layers.Lambda(lambda x: x * 200)
])


lr_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-8 * 10**(epoch / 20))

optimizer = tf.keras.optimizers.SGD(learning_rate=1e-8, momentum=0.9)

model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])

history = model.fit(dataset, epochs=100, callbacks=[lr_schedule])
```


30
25
20
15
10
5
0
10^{-8}
10^{-7}
10^{-6}
10^{-5}
10^{-4}

```
model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    tf.keras.layers.Conv1D(filters=32, kernel_size=5,
                           strides=1, padding="causal",
                           activation="relu")
    tf.keras.layers.LSTM(32, return_sequences=True),
    tf.keras.layers.LSTM(32),
    tf.keras.layers.Dense(1),
    tf.keras.layers.Lambda(lambda x: x * 200)
])


optimizer = tf.keras.optimizers.SGD(learning_rate=1e-5, momentum=0.9)

model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])

history = model.fit(dataset, epochs=500)
```

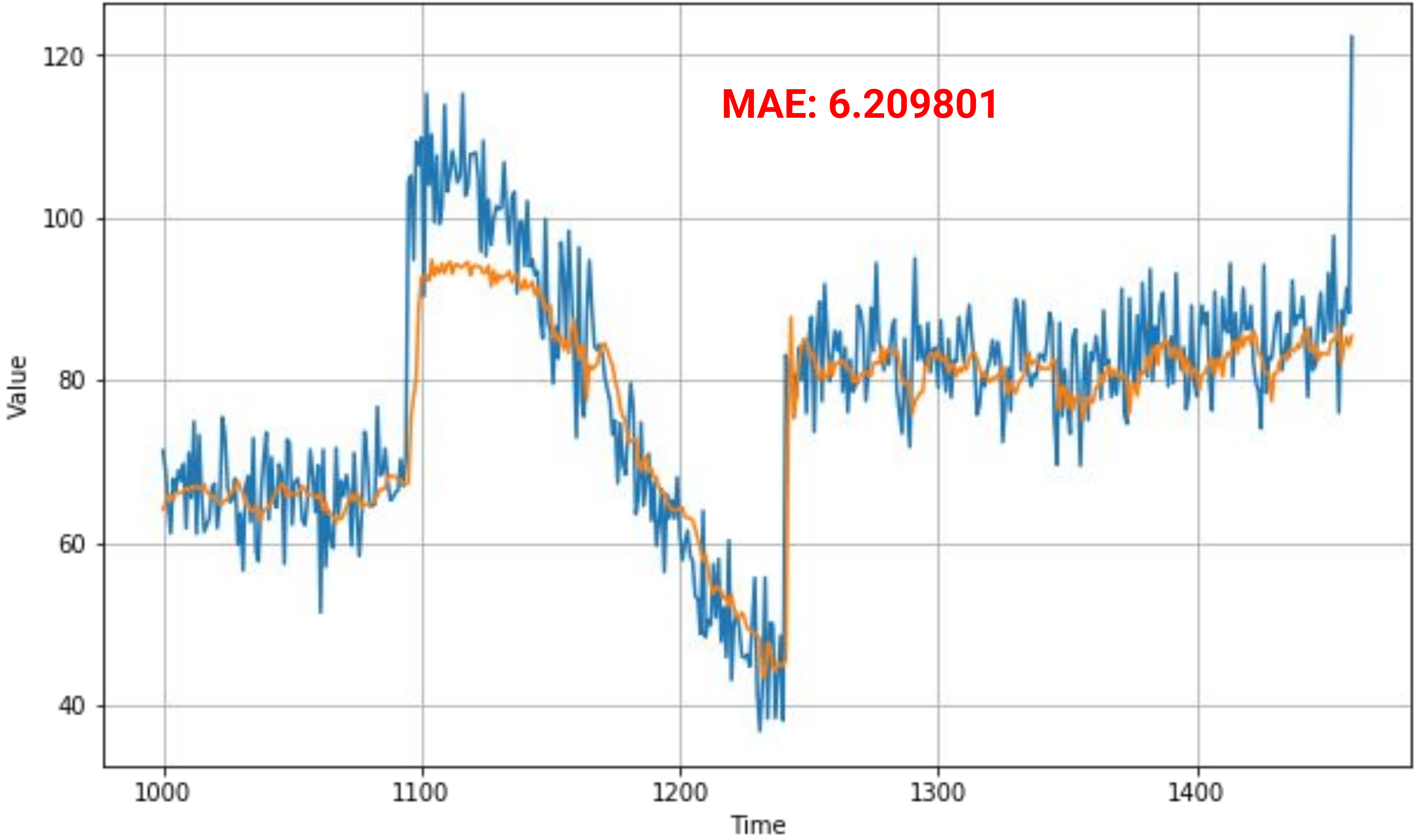

MAE: 4.985901
120
100
80
60
40
Value
1000
1100
1200
1300
1400
Time


MAE and Loss
MAE
Loss
Accuracy
Epochs
0
100
200
300
400
500
5
10
15
20
25

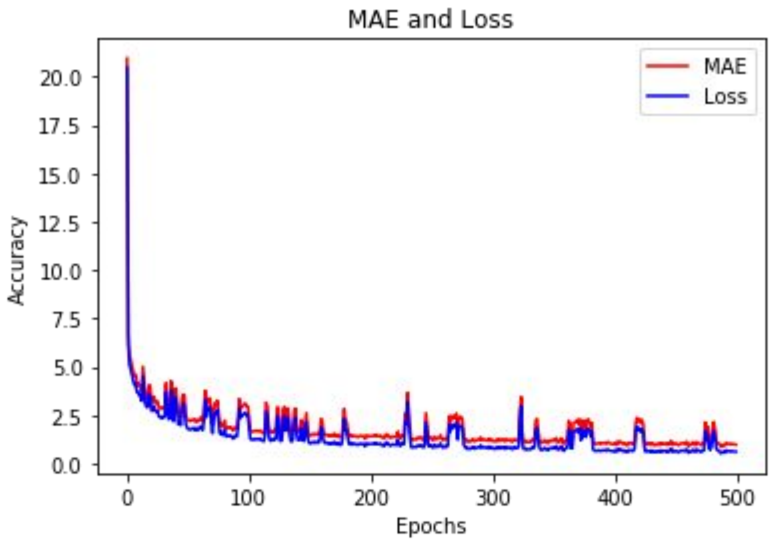
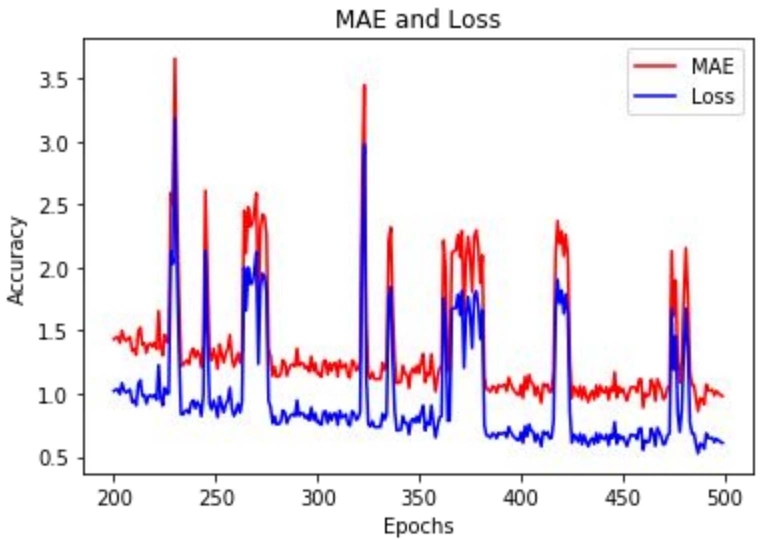

MAE and Loss
MAE
Loss
Accuracy
Epochs
200
250
300
350
400
450
500
3.9
4.0
4.1
4.2
4.3
4.4
4.5

```
model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    tf.keras.layers.Conv1D(filters=32, kernel_size=5,
                           strides=1, padding="causal",
                           activation="relu"),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32, return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(1),
    tf.keras.layers.Lambda(lambda x: x * 200)
])
```

```
Epoch 495/500
31/31 [==============================] - 2s 58ms/step - loss: 0.6491 - mae: 1.0314
Epoch 496/500
31/31 [==============================] - 2s 60ms/step - loss: 0.6155 - mae: 0.9857
Epoch 497/500
31/31 [==============================] - 2s 59ms/step - loss: 0.6425 - mae: 1.0207
Epoch 498/500
31/31 [==============================] - 2s 59ms/step - loss: 0.6330 - mae: 1.0046
Epoch 499/500
31/31 [==============================] - 2s 59ms/step - loss: 0.6155 - mae: 0.9877
Epoch 500/500
31/31 [==============================] - 2s 59ms/step - loss: 0.6111 - mae: 0.9806
```



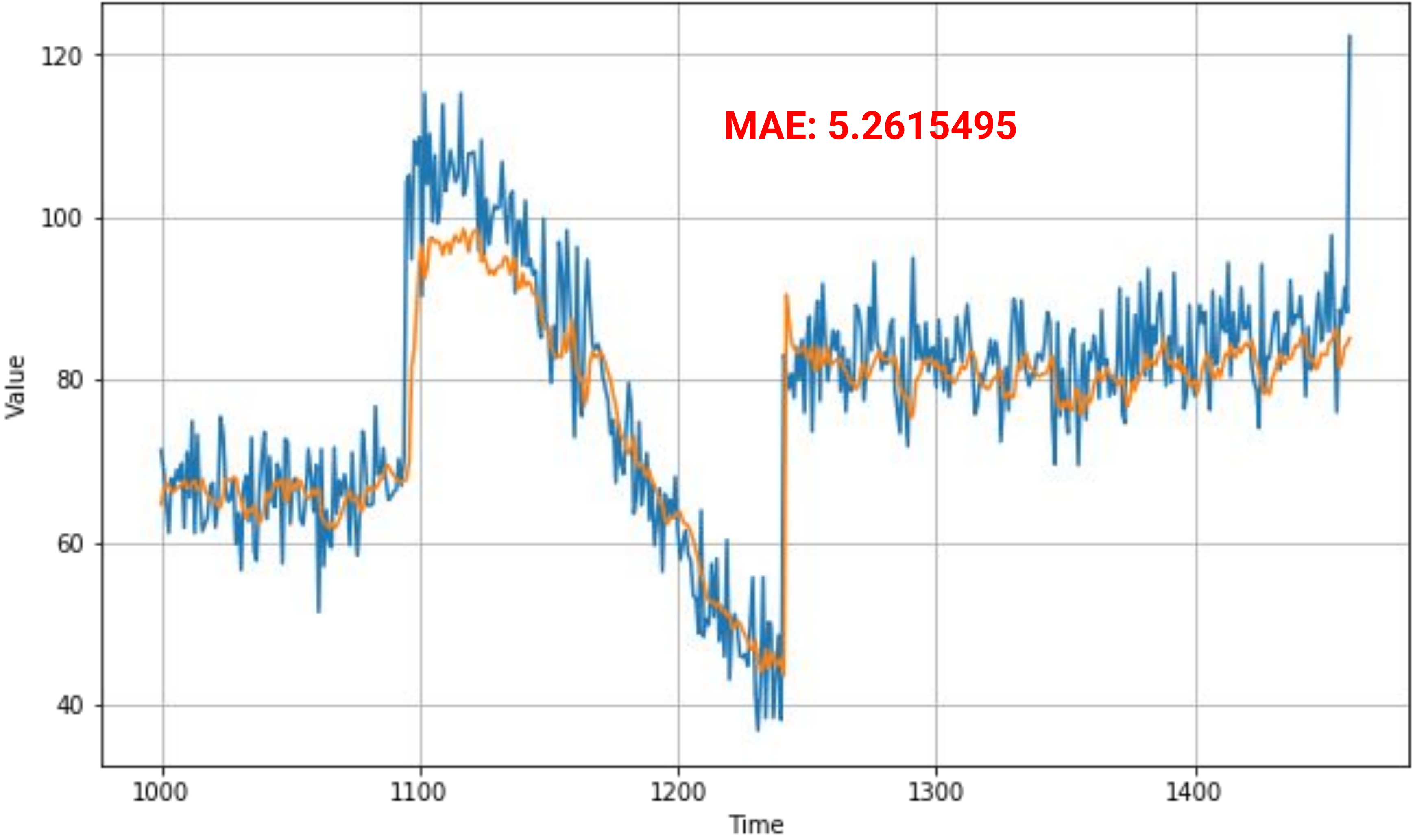
MAE: 6.209801
120
100
80
60
40
Value
1000
1100
1200
1300
1400
Time

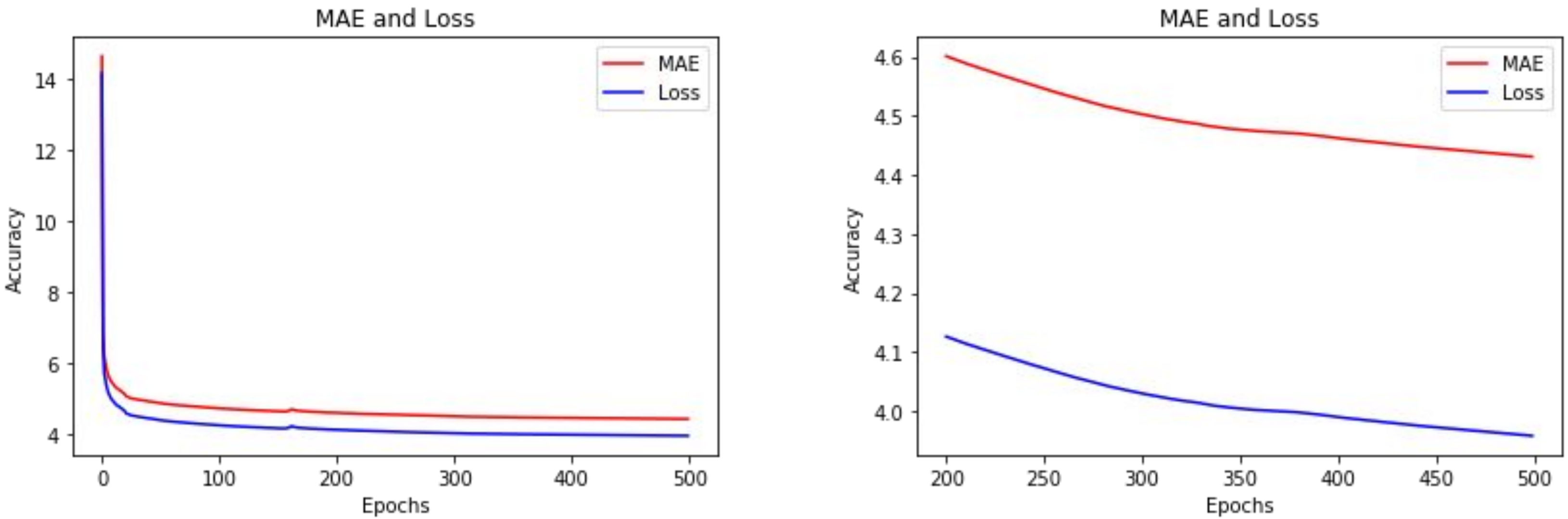

MAE and Loss
MAE
Loss
Accuracy
Epochs
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
100
200
300
400
500

MAE and Loss
MAE
Loss
Accuracy
Epochs
0.5
1.0
1.5
2.0
2.5
3.0
3.5
200
250
300
350
400
450
500

https://www.coursera.org/learn/deep-neural-network

youtube.com/watch?v=l4lSUAcvHF

Lecture 17.3 — Large Scale Machine Learning | Mini Batch Gradient Descent — [Andrew Ng]


MAE: 5.2615495
Value
Time
120
100
80
60
40
1000
1100
1200
1300
1400


MAE and Loss
MAE
Loss
Accuracy
Epochs
14
12
10
8
6
4
0
100
200
300
400
500
MAE and Loss
MAE
Loss
Accuracy
Epochs
4.6
4.5
4.4
4.3
4.2
4.1
4.0
200
250
300
350
400
450
500

https://www.kaggle.com/robervalt/sunspots

Dataset

Sunspots

Monthly Mean Total Sunspot Number - form 1749 to july 2018

Especuloide • updated a year ago (Version 3)

9

Data | Kernels (5) | Discussion | Activity | Metadata | Download (22 KB) | New Kernel

Usability 7.1 | **License** CC0: Public Domain | **Tags** No tags yet

Description

Context

Sunspots are temporary phenomena on the Sun's photosphere that appear as spots darker than the surrounding areas. They are regions of reduced surface temperature caused by concentrations of magnetic field flux that inhibit convection. Sunspots usually appear in pairs of opposite magnetic polarity. Their number varies according to the approximately 11-year solar cycle.

Source: https://en.wikipedia.org/wiki/Sunspot

Content :

Data (22 KB)

Data Sources	About this file	Columns
Sunspots.csv 3235 x 3	Sunspots - Monthly Mean Total Sunspot Number	#
		Date
		# Monthly Mean Total Sunspot Number

Sunspots.csv

```
,Date,Monthly Mean Total Sunspot Number
0,1749-01-31,96.7
1,1749-02-28,104.3
2,1749-03-31,116.7
3,1749-04-30,92.8
4,1749-05-31,141.7
5,1749-06-30,139.2
6,1749-07-31,158.0
7,1749-08-31,110.5
8,1749-09-30,126.5
9,1749-10-31,125.8
10,1749-11-30,264.3
11,1749-12-31,142.0
12,1750-01-31,122.2
13,1750-02-28,126.5
14,1750-03-31,148.7
15,1750-04-30,147.2
16,1750-05-31,150.0
17,1750-06-30,166.7
```

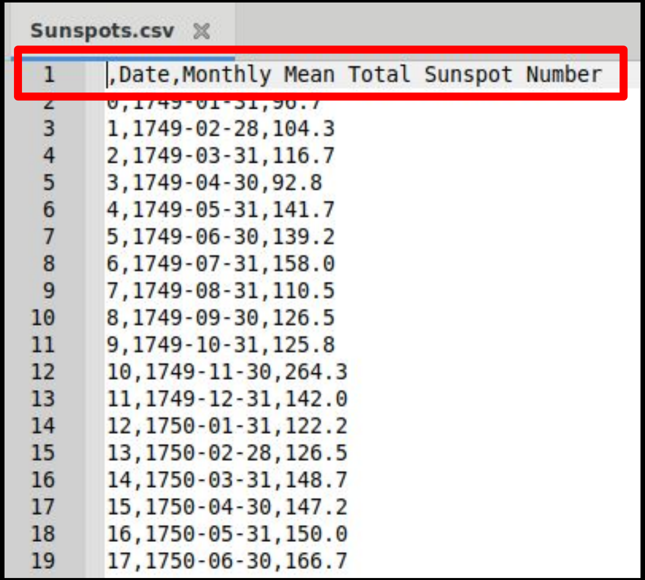
```
!wget --no-check-certificate \
    https://storage.googleapis.com/learning-datasets/Sunspots.csv \
    -O /tmp/sunspots.csv
```

```python
import csv
time_step = []
sunspots = []

with open('./sunspots.csv') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    next(reader)
    for row in reader:
        sunspots.append(float(row[2]))
        time_step.append(int(row[0]))
```

```
import csv
time_step = []
sunspots = []

with open('./sunspots.csv') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    next(reader)
    for row in reader:
        sunspots.append(float(row[2]))
        time_step.append(int(row[0]))
```

```
import csv
time_step = []
sunspots = []

with open('./sunspots.csv') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    next(reader)
    for row in reader:
        sunspots.append(float(row[2]))
        time_step.append(int(row[0]))
```

```python
import csv
time_step = []
sunspots = []

with open('./sunspots.csv') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    next(reader)
    for row in reader:
        sunspots.append(float(row[2]))
        time_step.append(int(row[0]))
```


Sunspots.csv

```
,Date,Monthly Mean Total Sunspot Number
0,1749-01-31,96.7
1,1749-02-28,104.3
2,1749-03-31,116.7
3,1749-04-30,92.8
4,1749-05-31,141.7
5,1749-06-30,139.2
6,1749-07-31,158.0
7,1749-08-31,110.5
8,1749-09-30,126.5
9,1749-10-31,125.8
10,1749-11-30,264.3
11,1749-12-31,142.0
12,1750-01-31,122.2
13,1750-02-28,126.5
14,1750-03-31,148.7
15,1750-04-30,147.2
16,1750-05-31,150.0
17,1750-06-30,166.7
```

```python
import csv
time_step = []
sunspots = []

with open('./sunspots.csv') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    next(reader)
    for row in reader:
        sunspots.append(float(row[2]))
        time_step.append(int(row[0]))
```

Sunspots.csv

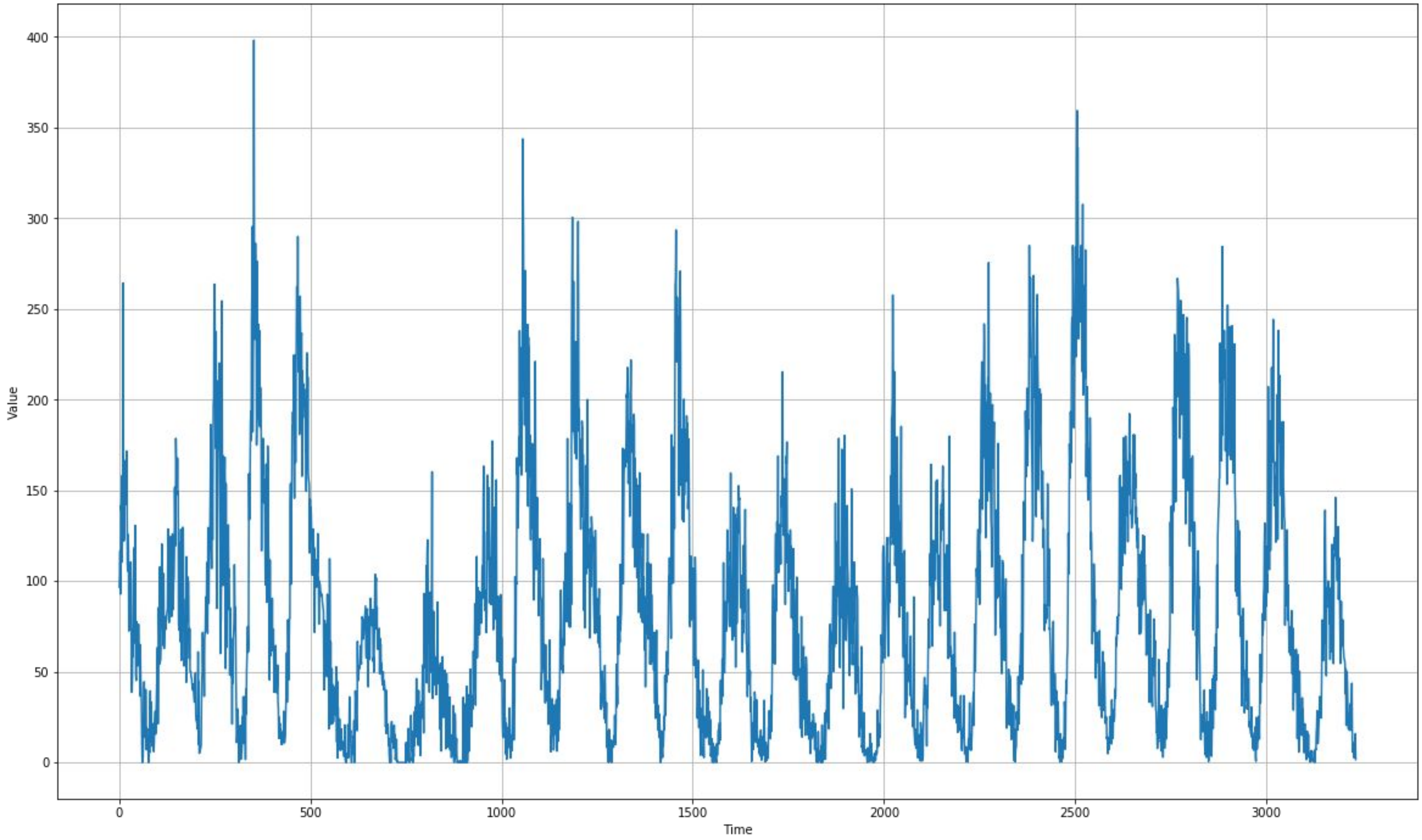
```
,Date,Monthly Mean Total Sunspot Number
0,1749-01-31,96.7
1,1749-02-28,104.3
2,1749-03-31,116.7
3,1749-04-30,92.8
4,1749-05-31,141.7
5,1749-06-30,139.2
6,1749-07-31,158.0
7,1749-08-31,110.5
8,1749-09-30,126.5
9,1749-10-31,125.8
10,1749-11-30,264.3
11,1749-12-31,142.0
12,1750-01-31,122.2
13,1750-02-28,126.5
14,1750-03-31,148.7
15,1750-04-30,147.2
16,1750-05-31,150.0
17,1750-06-30,166.7
```

```python
import csv
time_step = []
sunspots = []

with open('./sunspots.csv') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    next(reader)
    for row in reader:
        sunspots.append(float(row[2]))
        time_step.append(int(row[0]))
```

Sunspots.csv

```
,Date,Monthly Mean Total Sunspot Number
0,1749-01-31,96.7
1,1749-02-28,104.3
2,1749-03-31,116.7
3,1749-04-30,92.8
4,1749-05-31,141.7
5,1749-06-30,139.2
6,1749-07-31,158.0
7,1749-08-31,110.5
8,1749-09-30,126.5
9,1749-10-31,125.8
10,1749-11-30,264.3
11,1749-12-31,142.0
12,1750-01-31,122.2
13,1750-02-28,126.5
14,1750-03-31,148.7
15,1750-04-30,147.2
16,1750-05-31,150.0
17,1750-06-30,166.7
```

```
series = np.array(sunspots)
time = np.array(time_step)
```

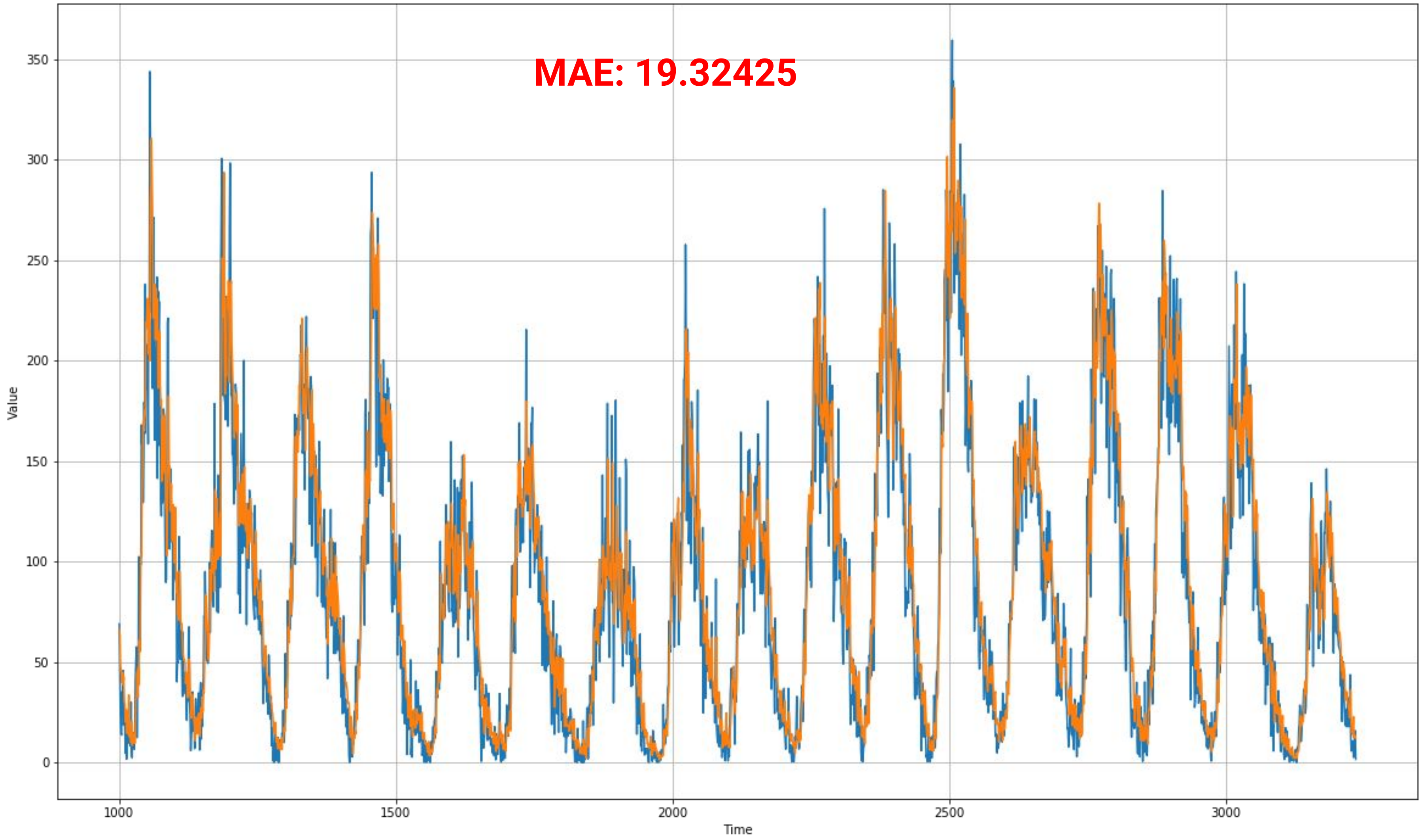


Value
Time
400
350
300
250
200
150
100
50
0
0
500
1000
1500
2000
2500
3000

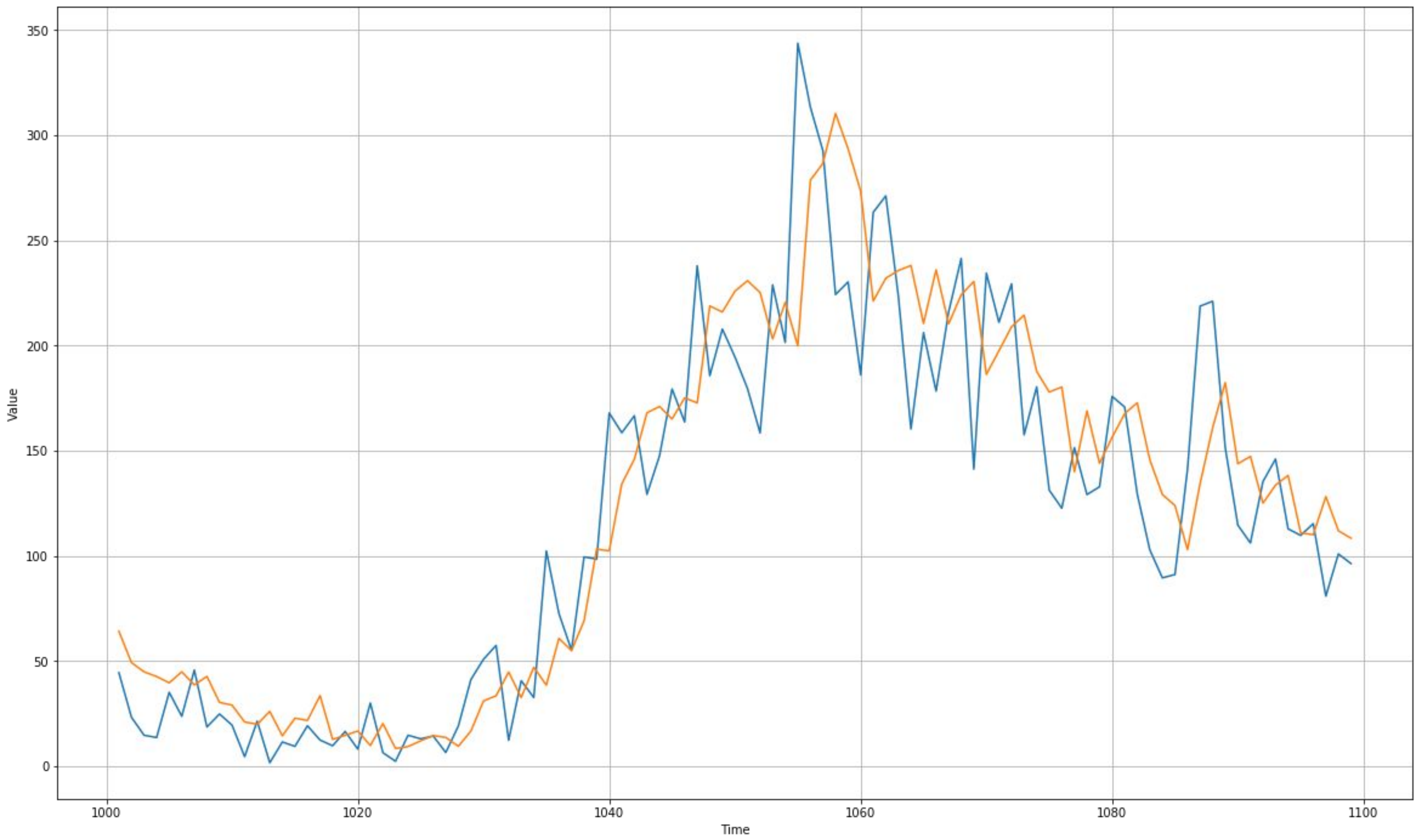
```
split_time = 1000
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]

window_size = 20
batch_size = 32
shuffle_buffer_size = 1000
```

```python
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
    dataset = tf.data.Dataset.from_tensor_slices(series)
    dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)
    dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))
    dataset = dataset.shuffle(shuffle_buffer)
    dataset = dataset.map(lambda window: (window[:-1], window[-1]))
    dataset = dataset.batch(batch_size).prefetch(1)
    return dataset
```

```python
dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size,)),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1, activation="relu")
])

model.compile(loss="mse", optimizer=tf.keras.optimizers.SGD(learning_rate=1e-6, momentum=0.9)

history = model.fit(dataset, epochs=100)
```

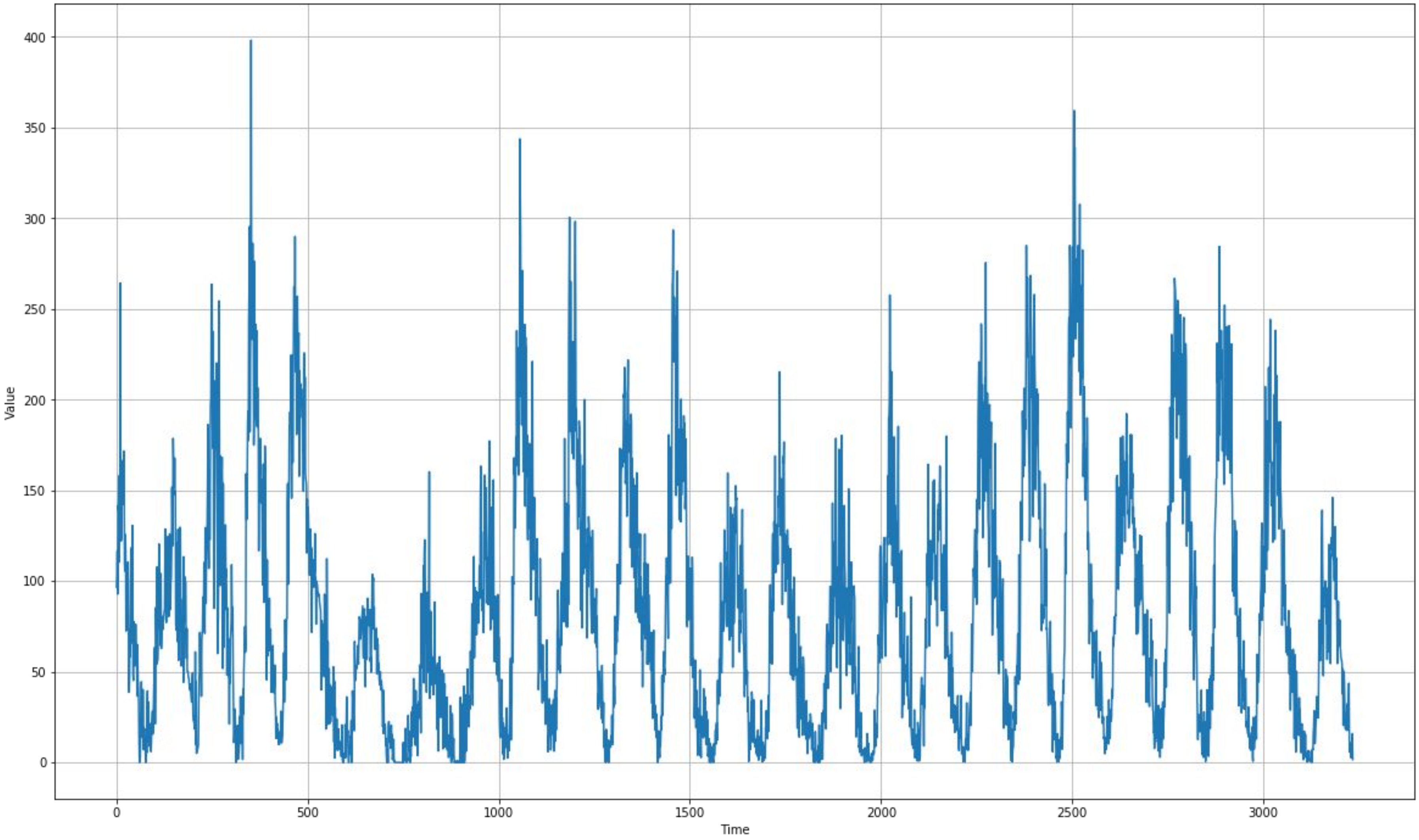


MAE: 19.32425
350
300
250
200
150
100
50
0
Value
1000
1500
2000
2500
3000
Time

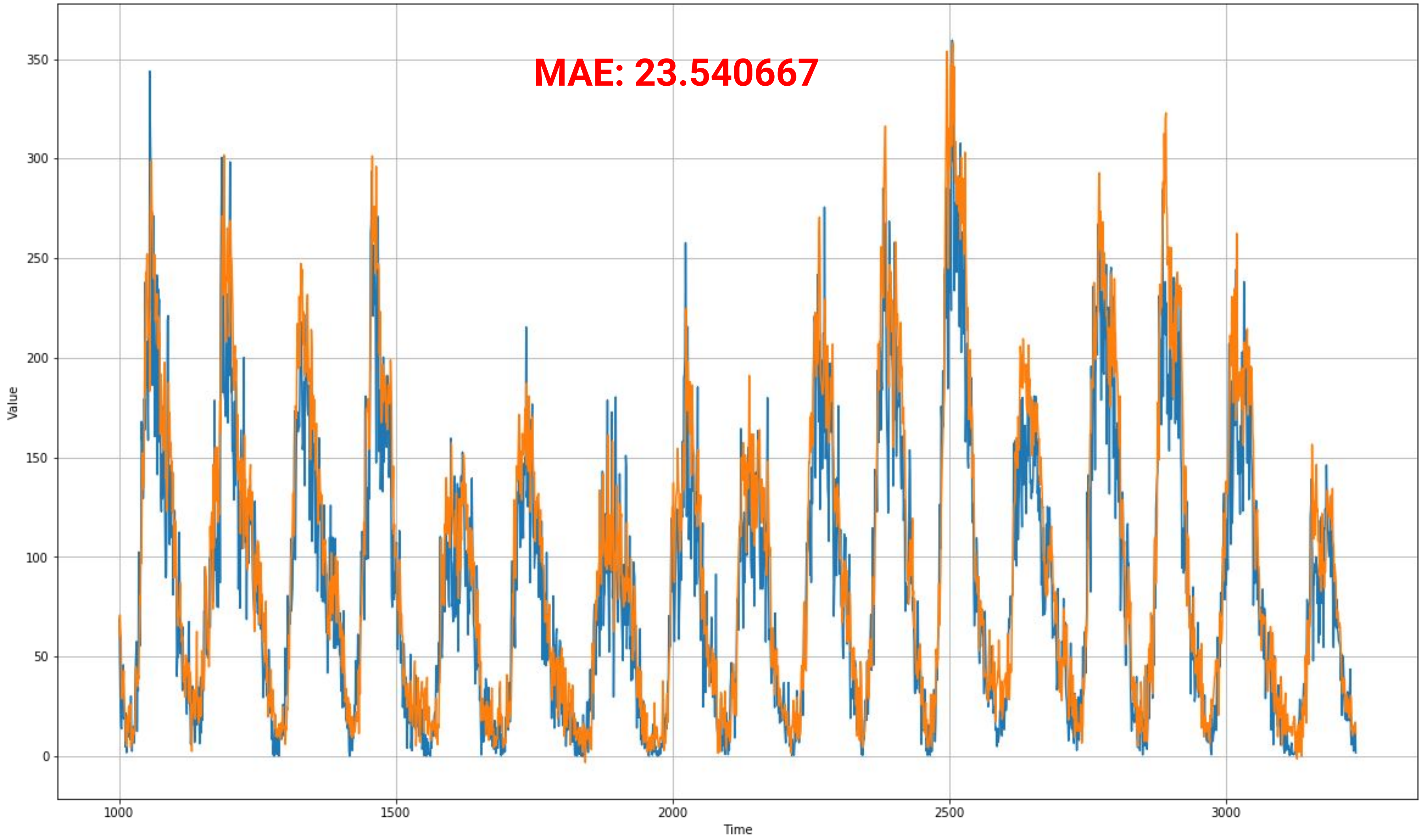


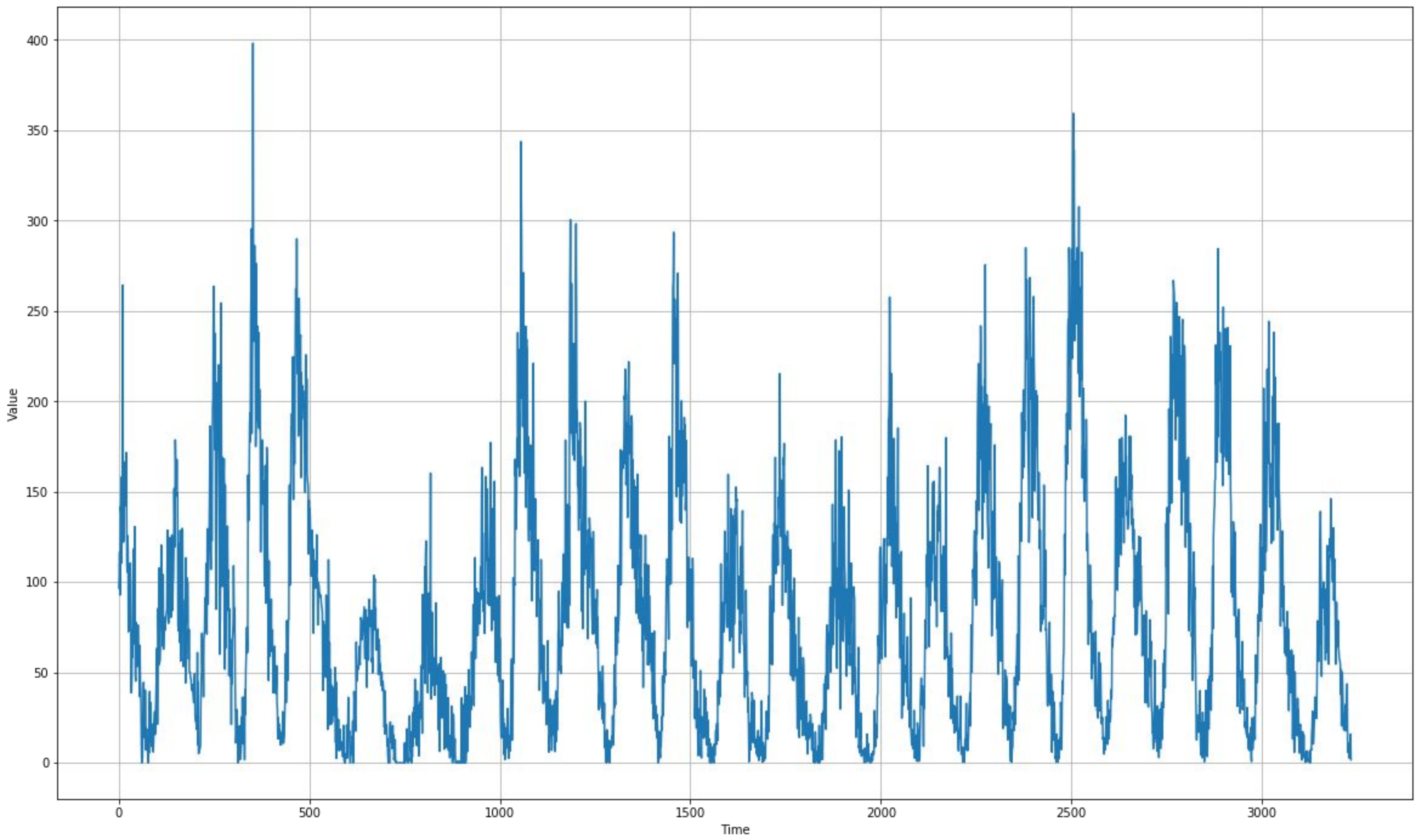
350
300
250
200
150
100
50
0
Value
1000
1020
1040
1060
1080
1100
Time

```
split_time = 1000
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]

window_size = 20
batch_size = 32
shuffle_buffer_size = 1000
```

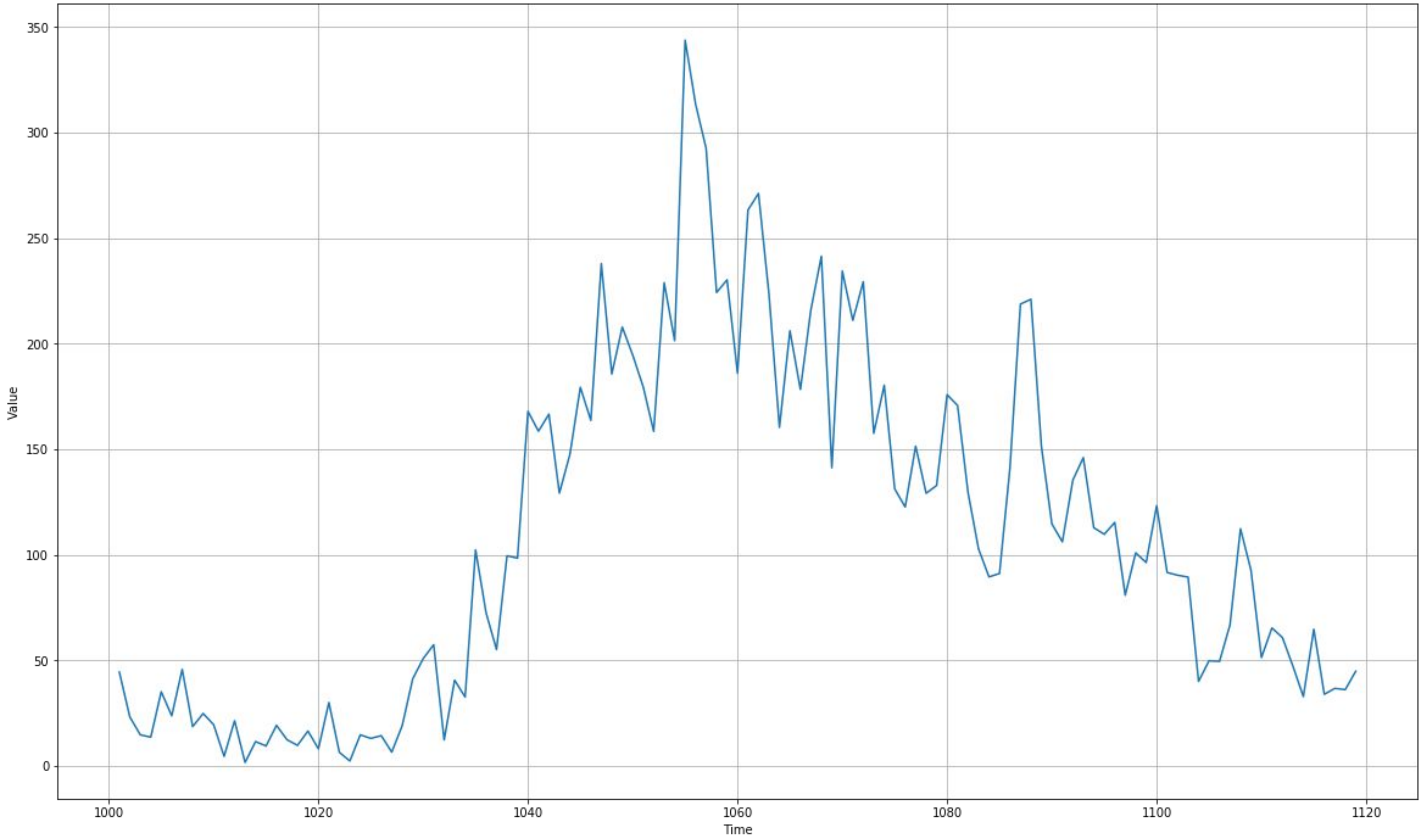

Value
Time
0
50
100
150
200
250
300
350
400
500
1000
1500
2000
2500
3000

```
split_time = 1000
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]

window_size = 132
batch_size = 32
shuffle_buffer_size = 1000
```


MAE: 23.540667
Value
Time
350
300
250
200
150
100
50
0
1000
1500
2000
2500
3000



400
350
300
250
200
150
100
50
0
Value
0
500
1000
1500
2000
2500
3000
Time



350
300
250
200
150
100
50
0
Value
1000
1020
1040
1060
1080
1100
1120
Time

```
split_time = 1000
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]

window_size = 30
batch_size = 32
shuffle_buffer_size = 1000
```

```
split_time = 1000
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]

window_size = 30
batch_size = 32
shuffle_buffer_size = 1000
```

```
split_time = 3000
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]

window_size = 30
batch_size = 32
shuffle_buffer_size = 1000
```

MAE: 15.1447315
250
200
150
100
50
0
Value
3000
3050
3100
3150
3200
Time

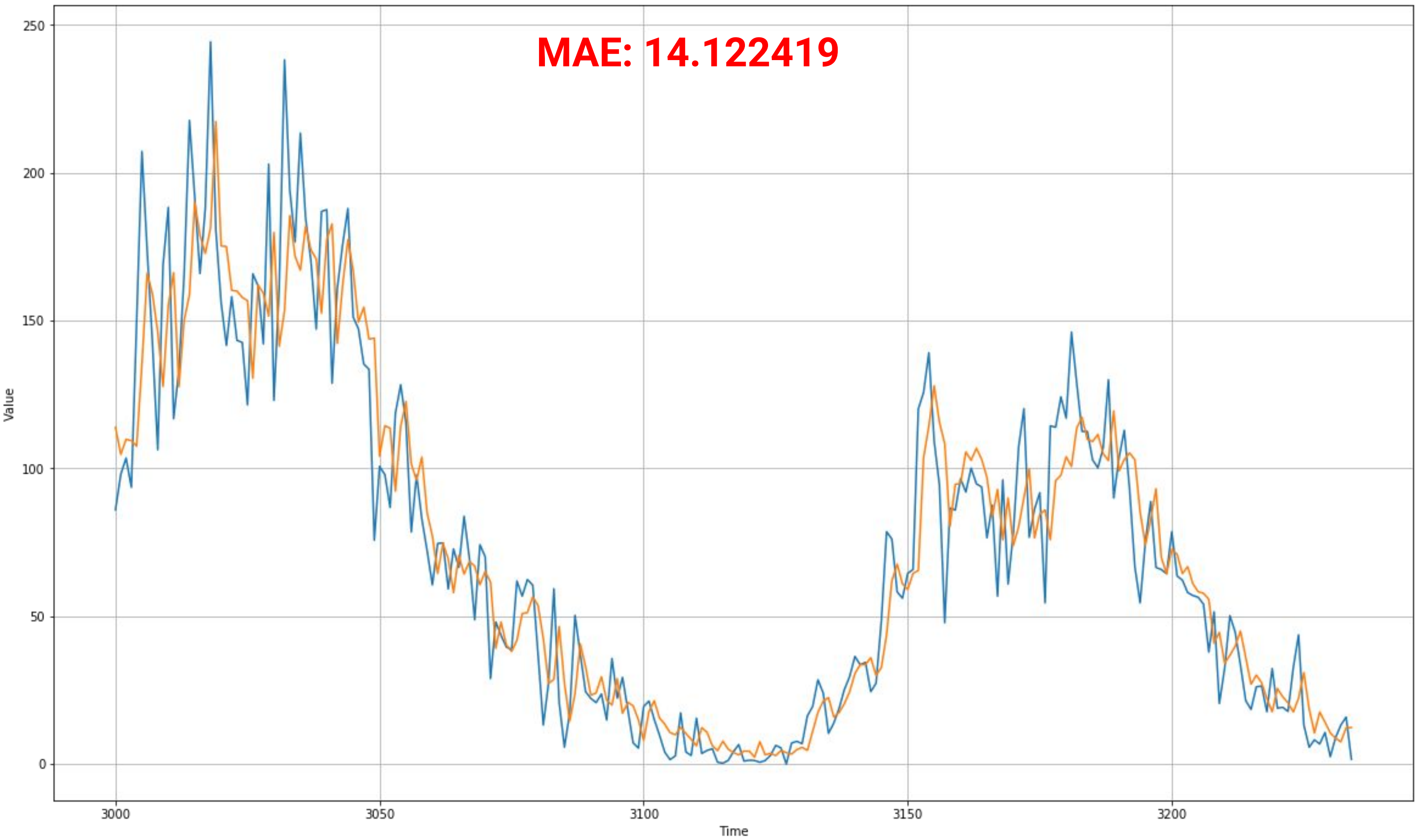
```
dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size,)),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1, activation="relu")
])

model.compile(loss="mse", optimizer=tf.keras.optimizers.SGD(learning_rate=1e-6, momentum=0.9))

history = model.fit(dataset, epochs=100)
```

```
dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size,)),
    tf.keras.layers.Dense(30, activation="relu"),
    tf.keras.layers.Dense(15, activation="relu"),
    tf.keras.layers.Dense(1, activation="relu")
])

model.compile(loss="mse", optimizer=tf.keras.optimizers.SGD(learning_rate=1e-6, momentum=0.9))

history = model.fit(dataset, epochs=100)
```


MAE: 14.348902
250
200
150
100
50
0
Value
3000
3050
3100
3150
3200
Time

```
dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size,)),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1, activation="relu")
])

model.compile(loss="mse", optimizer=tf.keras.optimizers.SGD(learning_rate=1e-6, momentum=0.9)

history = model.fit(dataset, epochs=100)
```

```
dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size,)),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1, activation="relu")
])

model.compile(loss="mse", optimizer=tf.keras.optimizers.SGD(learning_rate=1e-7, momentum=0.9)

history = model.fit(dataset, epochs=100)
```


MAE: 14.122419
250
200
150
100
50
0
Value
3000
3050
3100
3150
3200
Time

```
model.predict(series[3205:3235][np.newaxis])
```

```
model.predict(series[3205:3235][np.newaxis])

      7.0773993
```

https://www.sws.bom.gov.au/Solar/1/6

(last updated 01 Jun 2019 09:42 UT)

OBSERVED MONTHLY SUNSPOT NUMBERS

2001	142.6	121.5	165.8	161.7	142.1	202.9	123.0	161.5	238.2	194.1	176.6	213.4
2002	184.6	170.2	147.1	186.9	187.5	128.8	161.0	175.6	187.9	151.2	147.2	135.3
2003	133.5	75.7	100.7	97.9	86.8	118.7	128.3	115.4	78.5	97.8	82.9	72.2
2004	60.6	74.6	74.8	59.2	72.8	66.5	83.8	69.7	48.8	74.2	70.1	28.9
2005	48.1	43.5	39.6	38.7	61.9	56.8	62.4	60.5	37.2	13.2	27.5	59.3
2006	20.9	5.7	17.3	50.3	37.2	24.5	22.2	20.8	23.7	14.9	35.7	22.3
2007	29.3	18.4	7.2	5.4	19.5	21.3	15.1	9.8	4.0	1.5	2.8	17.3
2008	4.1	2.9	15.5	3.6	4.6	5.2	0.6	0.3	1.2	4.2	6.6	1.0
2009	1.3	1.2	0.6	1.2	2.9	6.3	5.5	0.0	7.1	7.7	6.9	16.3
2010	19.5	28.5	24.0	10.4	13.9	18.8	25.2	29.6	36.4	33.6	34.4	24.5
2011	27.3	48.3	78.6	76.1	58.2	56.1	64.5	65.8	120.1	125.7	139.1	109.3
2012	94.4	47.8	86.6	85.9	96.5	92.0	100.1	94.8	93.7	76.5	87.6	56.8
2013	96.1	60.9	78.3	107.3	120.2	76.7	86.2	91.8	54.5	114.4	113.9	124.2
2014	117.0	146.1	128.7	112.5	112.5	102.9	100.2	106.9	130.0	90.0	103.6	112.9
2015	93.0	66.7	54.5	75.3	88.8	66.5	65.8	64.4	78.6	63.6	62.2	58.0
2016	57.0	56.4	54.1	37.9	51.5	20.5	32.4	50.2	44.6	33.4	21.4	18.5
2017	26.1	26.4	17.7	32.3	18.9	19.2	17.8	32.6	43.7	13.2	5.7	8.2
2018	6.8	10.7	2.5	8.9	13.1	15.6	1.6	$\boxed{8.7}$	3.3	4.9	4.9	3.1
2019	7.8	0.8	9.5	9.1	10.1							

```python
split_time = 3000
window_size = 60

dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size,)),
    tf.keras.layers.Dense(20, activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1, activation="relu")
])

model.compile(loss="mse", optimizer=tf.keras.optimizers.SGD(learning_rate=1e-7, momentum=0.9))
```

```python
window_size = 60
batch_size = 64

dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)


model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    tf.keras.layers.Conv1D(filters=32, kernel_size=5, strides=1,
                           padding="causal", activation="relu")
    tf.keras.layers.LSTM(32, return_sequences=True),
    tf.keras.layers.LSTM(32),
    tf.keras.layers.Dense(30, activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1, activation="relu")
    tf.keras.layers.Lambda(lambda x: x * 400)
])


lr_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-8 * 10**(epoch / 20))
optimizer = tf.keras.optimizers.SGD(learning_rate=1e-8, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(), optimizer=optimizer, metrics=["mae"])
history = model.fit(dataset, epochs=100, callbacks=[lr_schedule])
```

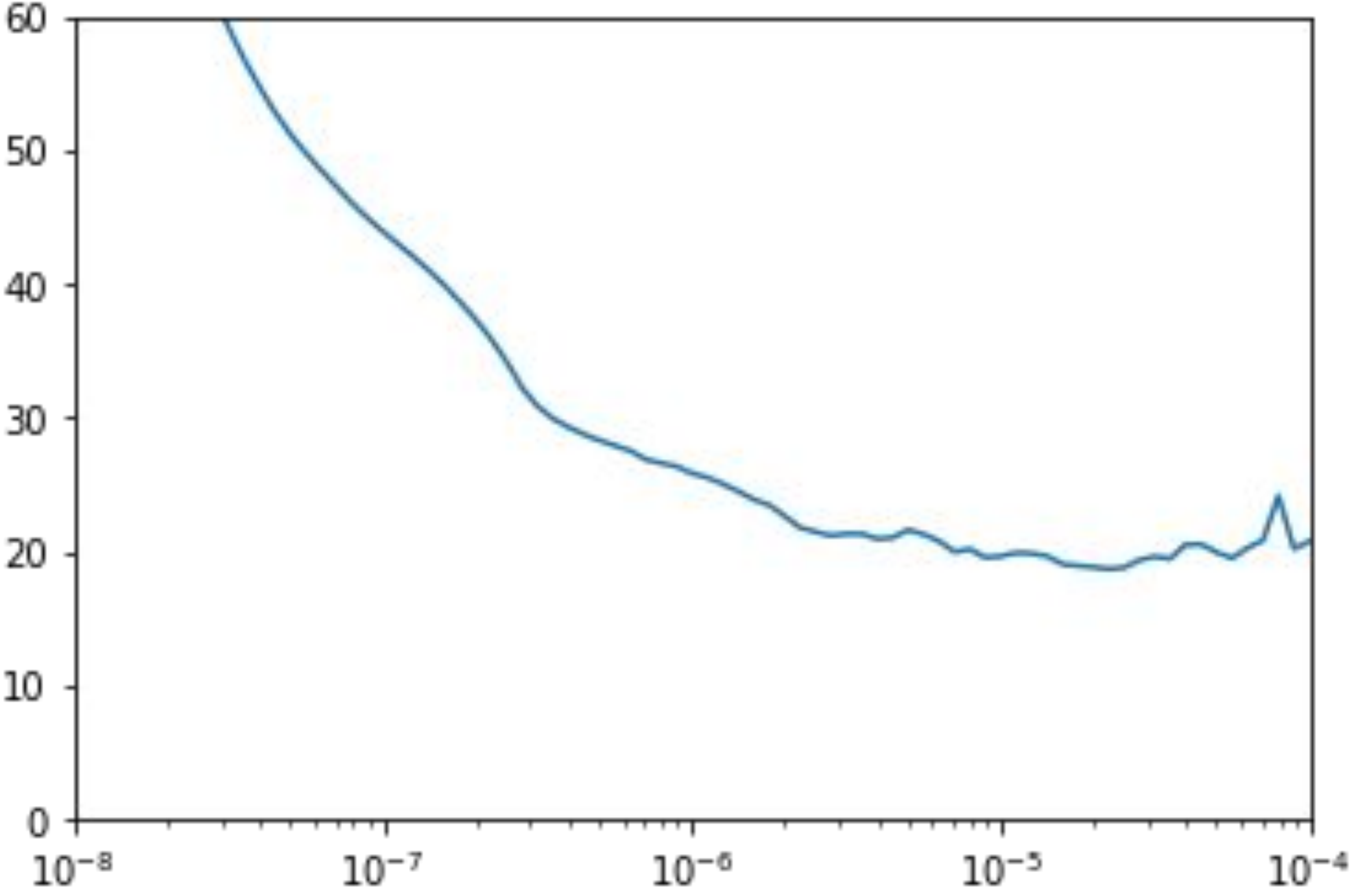
```
window_size = 60
batch_size = 64

dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    tf.keras.layers.Conv1D(filters=32, kernel_size=5, strides=1,
                           padding="causal", activation="relu")
    tf.keras.layers.LSTM(32, return_sequences=True),
    tf.keras.layers.LSTM(32),
    tf.keras.layers.Dense(30, activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1, activation="relu")
    tf.keras.layers.Lambda(lambda x: x * 400)
])


lr_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-8 * 10**(epoch / 20))
optimizer = tf.keras.optimizers.SGD(learning_rate=1e-8, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(), optimizer=optimizer, metrics=["mae"])
history = model.fit(dataset, epochs=100, callbacks=[lr_schedule])
```

```
window_size = 60
batch_size = 64

dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    tf.keras.layers.Conv1D(filters=32, kernel_size=5, strides=1,
                           padding="causal", activation="relu")
    tf.keras.layers.LSTM(32, return_sequences=True),
    tf.keras.layers.LSTM(32),
    tf.keras.layers.Dense(30, activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1, activation="relu")
    tf.keras.layers.Lambda(lambda x: x * 400)
])


lr_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-8 * 10**(epoch / 20))
optimizer = tf.keras.optimizers.SGD(learning_rate=1e-8, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(), optimizer=optimizer, metrics=["mae"])
history = model.fit(dataset, epochs=100, callbacks=[lr_schedule])
```

```python
window_size = 60
batch_size = 64

dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    tf.keras.layers.Conv1D(filters=32, kernel_size=5, strides=1,
                           padding="causal", activation="relu"),
    tf.keras.layers.LSTM(32, return_sequences=True),
    tf.keras.layers.LSTM(32),
    tf.keras.layers.Dense(30, activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1, activation="relu")
    tf.keras.layers.Lambda(lambda x: x * 400)
])


lr_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-8 * 10**(epoch / 20))
optimizer = tf.keras.optimizers.SGD(learning_rate=1e-8, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(), optimizer=optimizer, metrics=["mae"])
history = model.fit(dataset, epochs=100, callbacks=[lr_schedule])
```

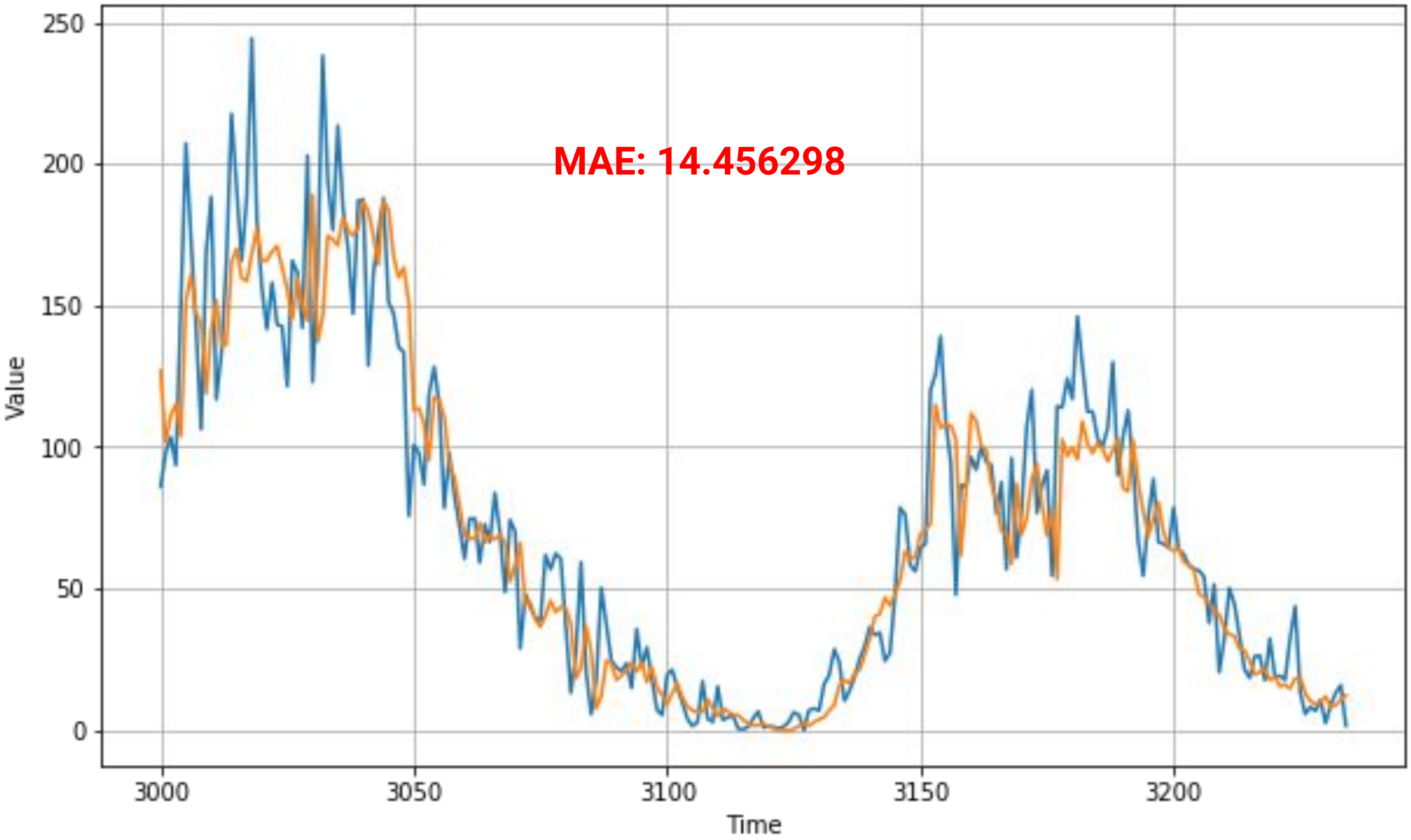
```
window_size = 60
batch_size = 64

dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    tf.keras.layers.Conv1D(filters=32, kernel_size=5, strides=1,
                           padding="causal", activation="relu")
    tf.keras.layers.LSTM(32, return_sequences=True),
    tf.keras.layers.LSTM(32),
    tf.keras.layers.Dense(30, activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1, activation="relu")
    tf.keras.layers.Lambda(lambda x: x * 400)
])


lr_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-8 * 10**(epoch / 20))
optimizer = tf.keras.optimizers.SGD(learning_rate=1e-8, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(), optimizer=optimizer, metrics=["mae"])
history = model.fit(dataset, epochs=100, callbacks=[lr_schedule])
```

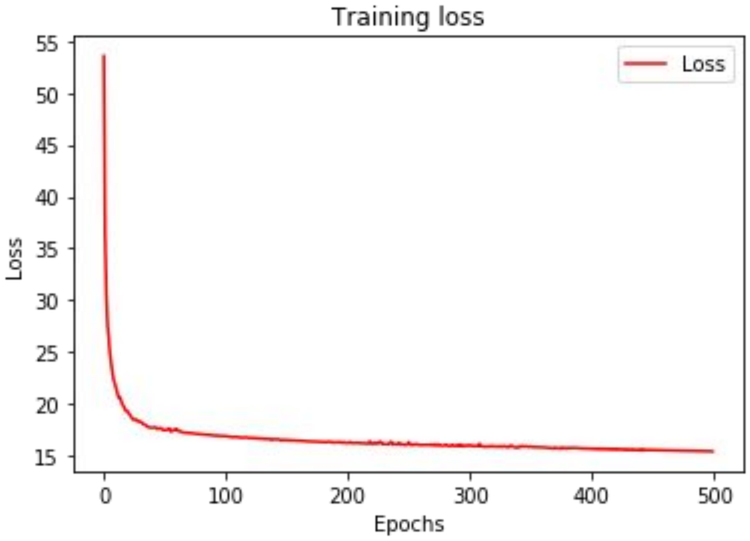
```python
window_size = 60
batch_size = 64

dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    tf.keras.layers.Conv1D(filters=32, kernel_size=5, strides=1,
                           padding="causal", activation="relu"),
    tf.keras.layers.LSTM(32, return_sequences=True),
    tf.keras.layers.LSTM(32),
    tf.keras.layers.Dense(30, activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1, activation="relu")
    tf.keras.layers.Lambda(lambda x: x * 400)
])


lr_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-8 * 10**(epoch / 20))
optimizer = tf.keras.optimizers.SGD(learning_rate=1e-8, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(), optimizer=optimizer, metrics=["mae"])
history = model.fit(dataset, epochs=100, callbacks=[lr_schedule])
```

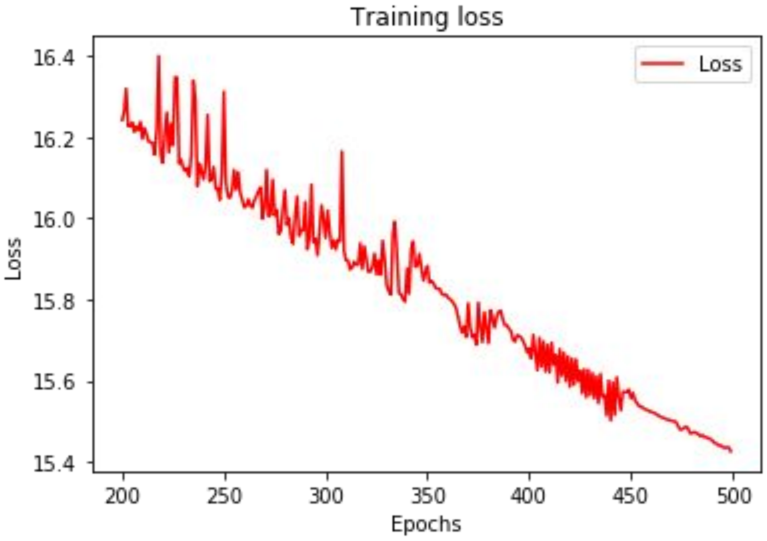


60
50
40
30
20
10
0
10^{-8}
10^{-7}
10^{-6}
10^{-5}
10^{-4}

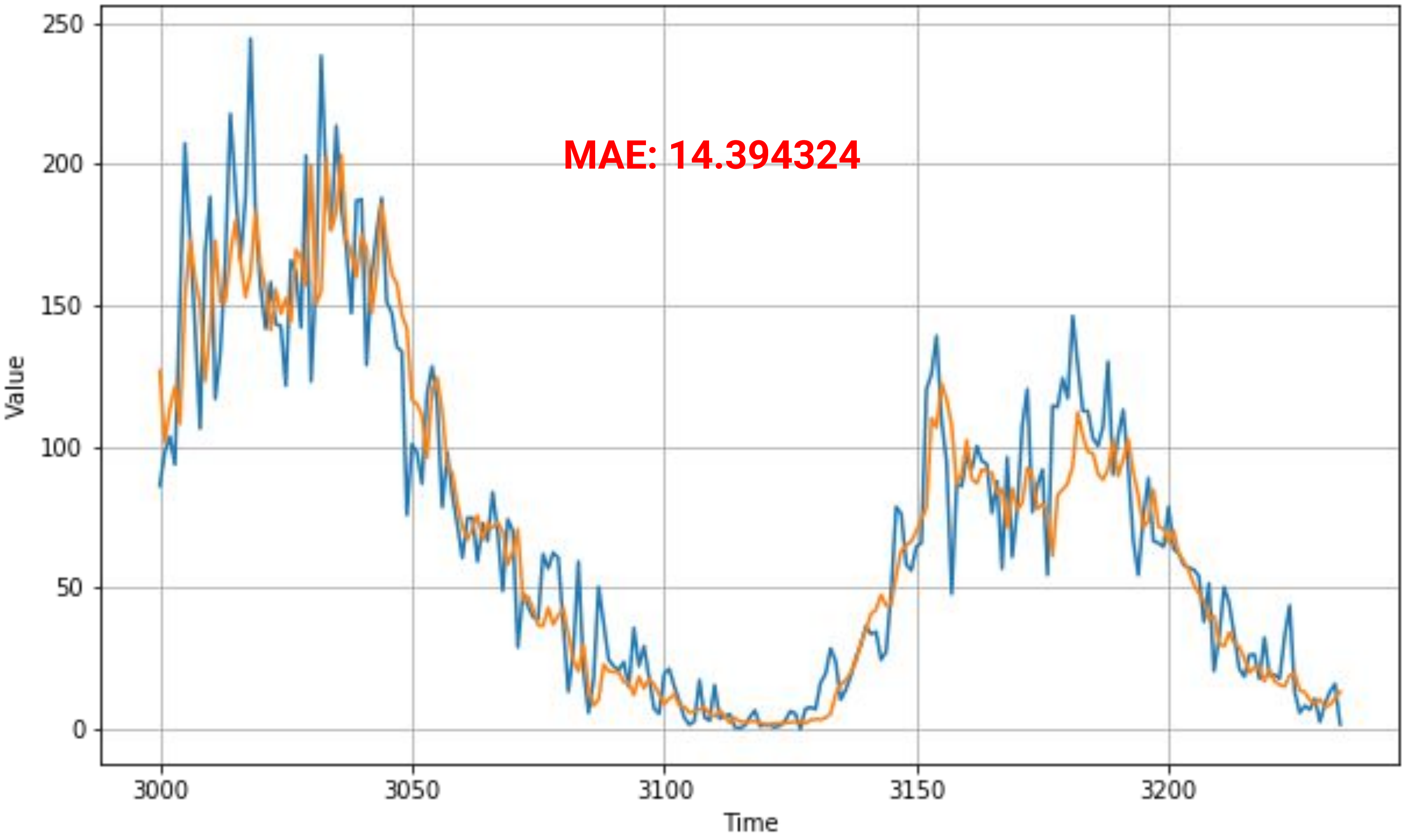
```
window_size = 60
batch_size = 64

dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    tf.keras.layers.Conv1D(filters=32, kernel_size=5, strides=1,
                           padding="causal", activation="relu")
    tf.keras.layers.LSTM(32, return_sequences=True),
    tf.keras.layers.LSTM(32),
    tf.keras.layers.Dense(30, activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1, activation="relu")
    tf.keras.layers.Lambda(lambda x: x * 400)
])


optimizer = tf.keras.optimizers.SGD(learning_rate=1e-5, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(), optimizer=optimizer, metrics=["mae"])
history = model.fit(dataset, epochs=500)
```

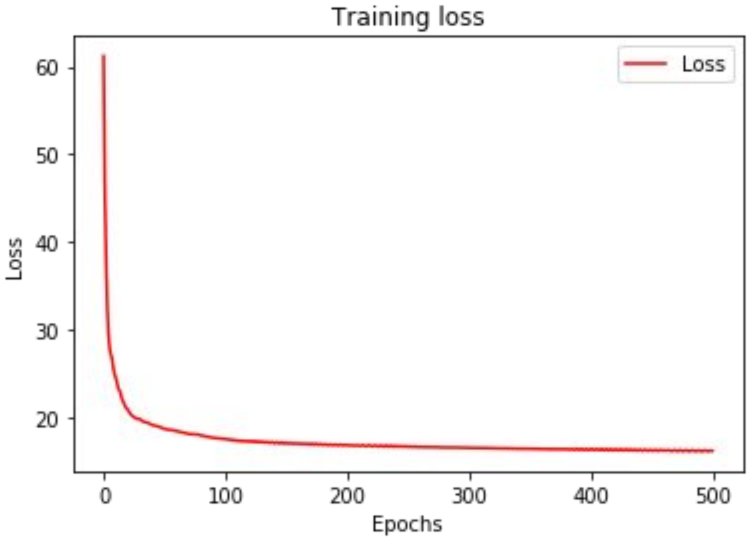

MAE: 14.456298
Value
Time
250
200
150
100
50
0
3000
3050
3100
3150
3200

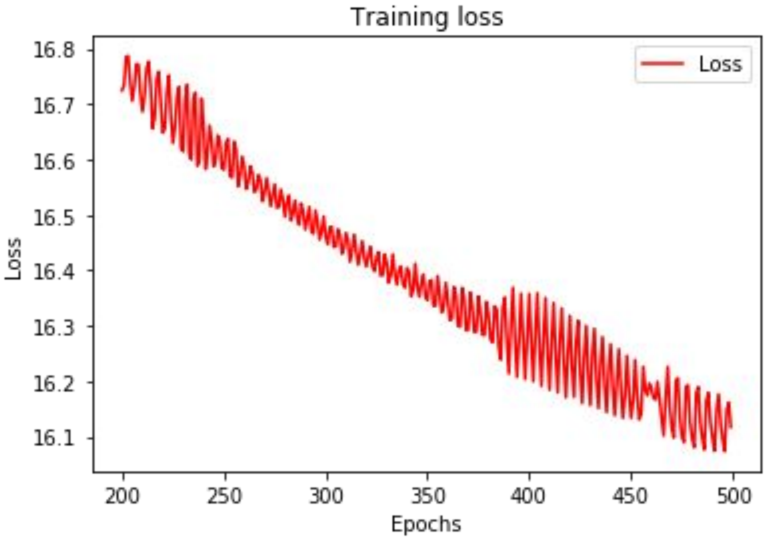

Training loss
Loss
Epochs
Loss


Training loss
Loss
Epochs
Loss

```
window_size = 60
batch_size = 256

dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    tf.keras.layers.Conv1D(filters=32, kernel_size=5, strides=1,
                           padding="causal", activation="relu")
    tf.keras.layers.LSTM(32, return_sequences=True),
    tf.keras.layers.LSTM(32),
    tf.keras.layers.Dense(30, activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1, activation="relu")
    tf.keras.layers.Lambda(lambda x: x * 400)
])


optimizer = tf.keras.optimizers.SGD(learning_rate=1e-5, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(), optimizer=optimizer, metrics=["mae"])
history = model.fit(dataset, epochs=500)
```


MAE: 14.394324
Value
Time
250
200
150
100
50
0
3000
3050
3100
3150
3200

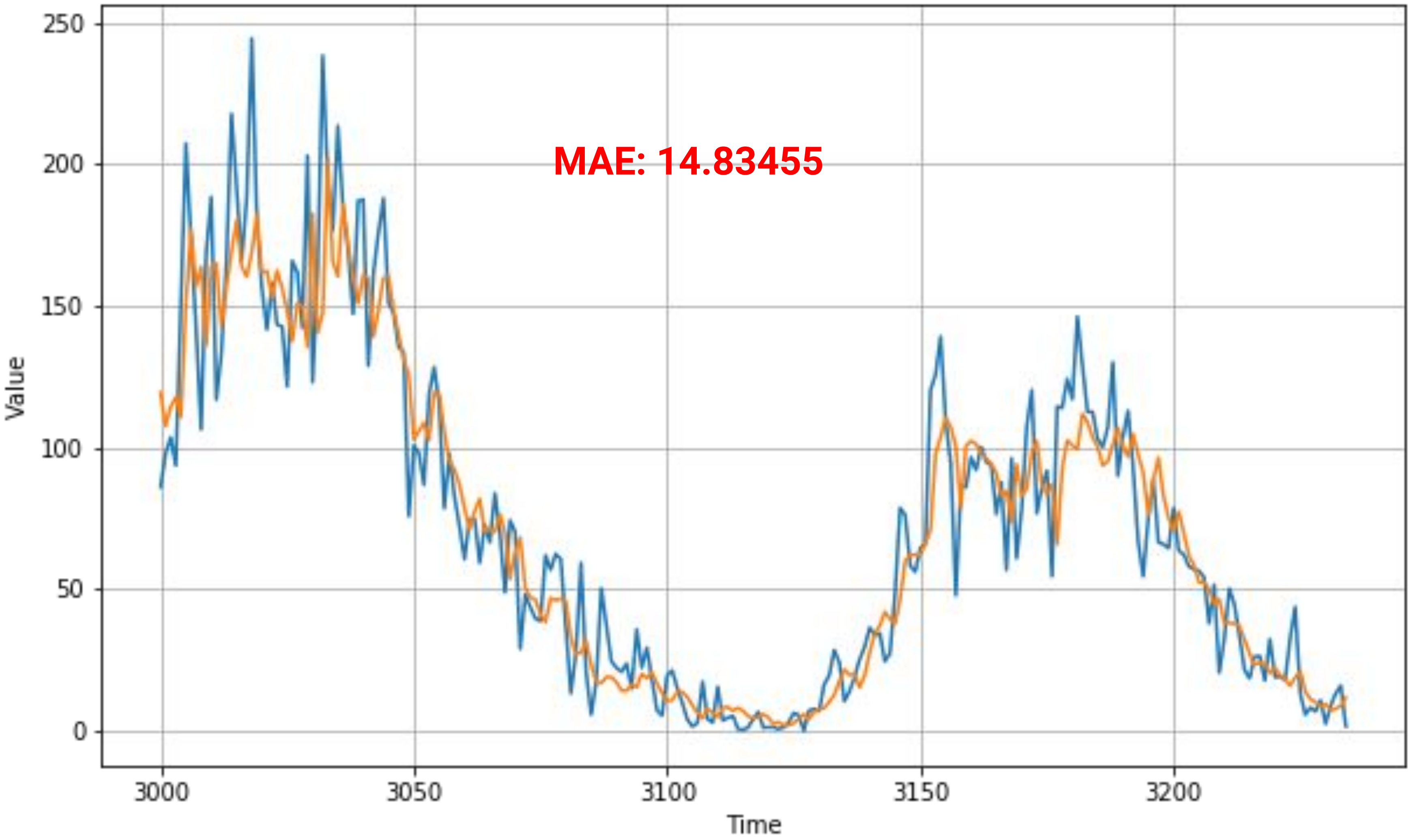

Training loss
Loss
60
50
40
30
20
0
100
200
300
400
500
Epochs


Training loss
Loss
16.8
16.7
16.6
16.5
16.4
16.3
16.2
16.1
200
250
300
350
400
450
500
Epochs

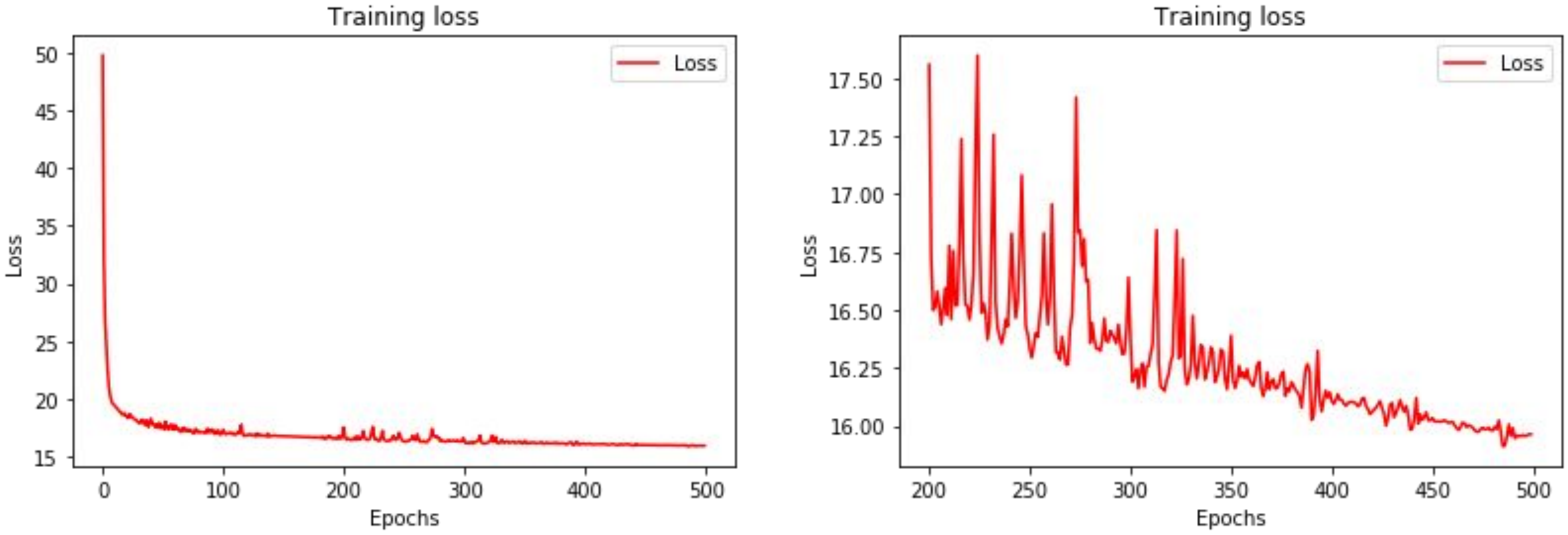
```
window_size = 60
batch_size = 250

dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    tf.keras.layers.Conv1D(filters=60, kernel_size=5, strides=1,
                           padding="causal", activation="relu")
    tf.keras.layers.LSTM(60, return_sequences=True),
    tf.keras.layers.LSTM(60),
    tf.keras.layers.Dense(30, activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1, activation="relu")
    tf.keras.layers.Lambda(lambda x: x * 400)
])


optimizer = tf.keras.optimizers.SGD(learning_rate=1e-5, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(), optimizer=optimizer, metrics=["mae"])
history = model.fit(dataset, epochs=500)
```

```
window_size = 60
batch_size = 250

dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    tf.keras.layers.Conv1D(filters=60, kernel_size=5, strides=1,
                        padding="causal", activation="relu")
    tf.keras.layers.LSTM(60, return_sequences=True),
    tf.keras.layers.LSTM(60),
    tf.keras.layers.Dense(30, activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1, activation="relu")
    tf.keras.layers.Lambda(lambda x: x * 400)
])


optimizer = tf.keras.optimizers.SGD(learning_rate=1e-5, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(), optimizer=optimizer, metrics=["mae"])
history = model.fit(dataset, epochs=500)
```

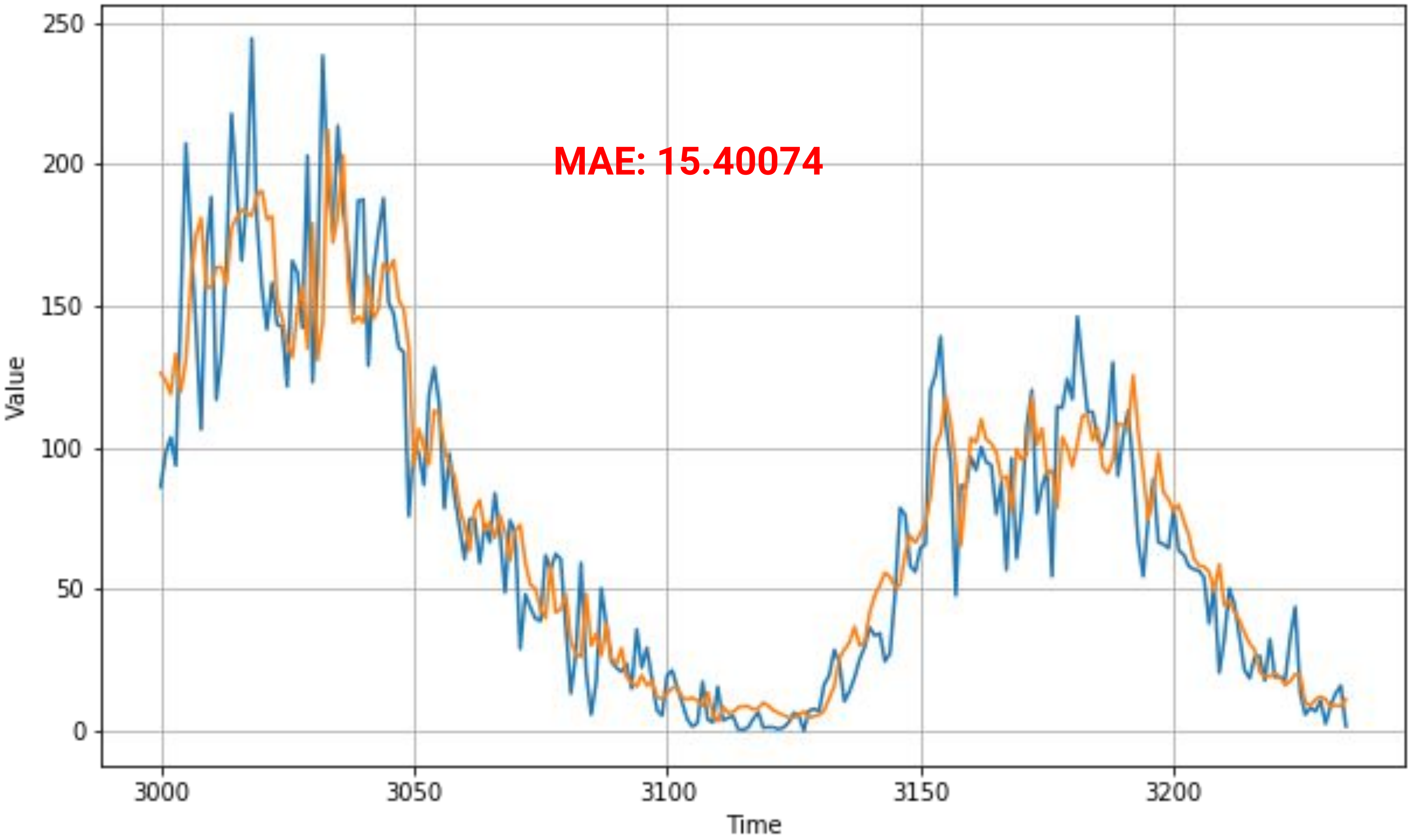
```
window_size = 60
batch_size = 250

dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    tf.keras.layers.Conv1D(filters=60, kernel_size=5, strides=1,
                           padding="causal", activation="relu")
    tf.keras.layers.LSTM(60, return_sequences=True),
    tf.keras.layers.LSTM(60),
    tf.keras.layers.Dense(30, activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1, activation="relu")
    tf.keras.layers.Lambda(lambda x: x * 400)
])


optimizer = tf.keras.optimizers.SGD(learning_rate=1e-5, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(), optimizer=optimizer, metrics=["mae"])
history = model.fit(dataset, epochs=500)
```

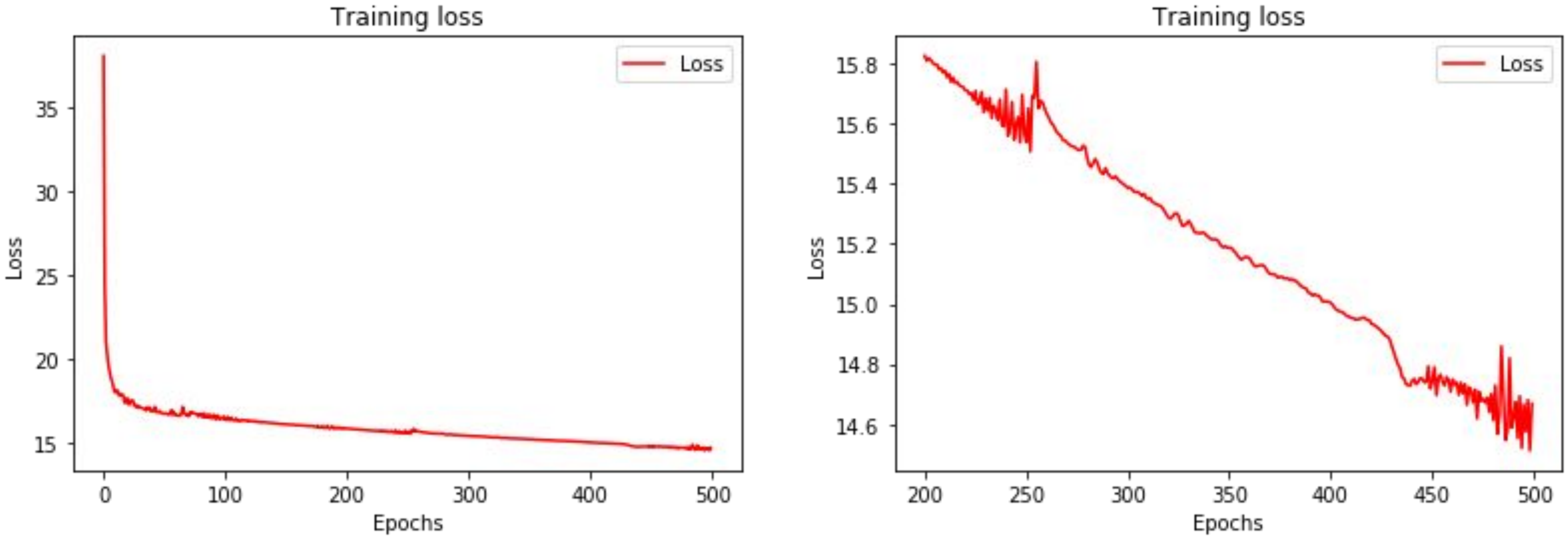


MAE: 14.83455
250
200
150
100
50
0
Value
3000
3050
3100
3150
3200
Time



Training loss
Loss
Epochs
Loss
Training loss
Loss
Epochs
Loss


MAE: 15.40074
Value
Time
250
200
150
100
50
0
3000
3050
3100
3150
3200


Training loss
Loss
Epochs
0
100
200
300
400
500
15
20
25
30
35
Training loss
Loss
Epochs
200
250
300
350
400
450
500
14.6
14.8
15.0
15.2
15.4
15.6
15.8