

$$F = G \frac{m_1 m_2}{d^2}$$

# PREDIKSI NILAI GRAFIK COSINUS DENGAN KERAS TENSORFLOW

GREGORIUS PRADANA SATRIAWAN - 2101201041

$$\frac{df}{dt} = \lim_{h \rightarrow 0} \frac{f(t+h) - f(t)}{h}$$

# PENDAHULUAN

- Deep Learning (DL) merupakan salah satu jenis turunan Machine Learning (ML) dan Artificial Intellegent (AI).
- Dengan menggunakan model DL/ML, kita bisa melakukan pemrosesan data, seperti klasifikasi maupun prediksi.
- Pada slide ini, saya akan menjelaskan teknik DL/ML untuk melakukan prediksi nilai sederhana, yakni prediksi nilai grafik cosinus, dalam model yang telah dibuat sebelumnya.
- Adapun model tersebut dapat dilihat dalam github (<https://github.com/griops/Prediksi-Grafik-Cos>).
- Model dibuat menggunakan Keras, yakni API dari TensorFlow, yang digunakan dalam deep learning.
- Adapun TensorFlow merupakan library machine learning yang bersifat open source.
- Program dijalankan secara cloud menggunakan Google Colabs.
- Dilakukan evaluasi model menggunakan nilai loss dan mean square error (MSE).
- Pada kesempatan ini, dilakukan evaluasi perbandingan jumlah sampel data, terhadap prediksi grafik cosinus, dilihat dari parameter loss dan MSE.

# LANGKAH Pengerjaan

- Import Dependency
- Pembuatan Dataset
  - Membuat Data Cosinus
  - Membuat 'Noise' pada Data
  - Pemecahan Dataset
- Pembuatan Model
  - Membuat Desain Model
  - Melakukan Training
- Evaluasi
  - Plotting Parameter Output
  - Evaluasi Hasil

# IMPORT DEPENDENCY

- Dilakukan pendefinisian library yang akan digunakan.

```
# Pendefinisian Lokasi Penyimpanan Model yang akan dibuat
import os
MODELS_DIR = 'models/'
if not os.path.exists(MODELS_DIR):
    os.mkdir(MODELS_DIR)
MODEL_TF = MODELS_DIR + 'model'
```

```
# Memanggil Library yang akan digunakan : Tensor Flow, Keras, Numpy, Panda, Matplotlib, Math

# Memanggil TensorFlow, yang digunakan sebagai library machine learning. (TensorFlow bersifat open source)
import tensorflow as tf

# Memanggil Keras, yakni sebuah API dari TensorFlow, yang digunakan dalam deep learning.
from tensorflow import keras

# Memanggil Numpy, sebuah library untuk pengolahan matematis.
import numpy as np

# Memanggil Pandas, sebuah library untuk pengolahan data.
import pandas as pd

# Memanggil Matplotlib, untuk membuat grafik.
import matplotlib.pyplot as plt

# Memanggil Math, yakni library Python untuk fngsi matematis
import math

# Melakukan pengaturan 'seed', untuk generasi angka acak di numpy maupun tensor flow
seed = 1
np.random.seed(seed)
tf.random.set_seed(seed)
```

Kita akan membuat angka acak, dengan nilai `x`, lalu kita akan menghitung nilai cosinus dari 'x' tersebut, dan memvisualisaikannya dengan sebuah grafik

```
# Jumlah sampel angka. Kita akan membuat grafik fungsi cos, dengan beberapa angka yang acak. Disini kita melakukan generasi beberapa nilai angka.  
# Kita melakukan perubahan jumlah sampel angka yang akan dibuat menjadi grafik, yakni dengan sampel 100 titik, 500 titik dan 1000 titik  
  
SAMPLES = 500  
# Jika akan mengganti jumlah titik, dilakukan dengan mengganti angka SAMPLES = 100 atau 1000  
  
# Membuat data dengan distribusi uniform, berdasarkan angka acak, dengan nilai 0 hingga  $2\pi$ , yang merepresentasikan 1 gelombang nilai cos.  
x_values = np.random.uniform(low=0, high=2*math.pi, size=SAMPLES).astype(np.float32)  
  
# Nilai diacak supaya tidak berurutan.  
np.random.shuffle(x_values)  
  
# Menghitung nilai cos(x), yang akan di plot menjadi sumbu y grafik  
y_values = np.cos(x_values).astype(np.float32)  
  
# Melakukan plot data ke grafik, kita lihat apakah yang akan terjadi, seharusnya muncul grafik fungsi cos, dengan titik-titik warna hijau (green g.)  
plt.plot(x_values, y_values, 'g.')  
plt.show()
```

## PEMBUATAN DATASET

- Membuat Data Cosinus



# PEMBUATAN DATASET

## MENAMBAHKAN 'NOISE' PADA DATA

Dengan generasi fungsi cos, kita telah melihat grafik yang cukup bagus.

Kita akan mencoba 'merusak' grafik dengan mengacak nilai pada sumbu y. Seperti pada data dalam kenyataannya, kadang kita mendapat data yang sedikit rusak atau bergeser nilainya. Kita menambahkan 'noise' secara acak, dapat dicermati hasilnya pada grafik berikutnya :

```
▶ # Add a small random number to each y value
y_values += 0.15 * np.random.randn(*y_values.shape)

# Plot our data
plt.plot(x_values, y_values, 'g.')
plt.show()

#Kita lihat jika data menjadi teracak 15% dari kondisi normal, dapat dilihat seperti grafik di bawah:
```

# PEMBUATAN DATASET

Kita sekarang memiliki dataset yang sedikit acak (terkena 'noise')

Untuk mengevaluasi model yang akan kita buat, kita perlu membandingkan prediksi nilai terhadap data aslinya. Lalu kita cek, apakah cocok. Evaluasi ini berlangsung ketika training model 'validasi data', maupun setelah training 'data testing'. Perlu digarisbawahi bahwa kita menggunakan data yang berbeda, untuk training, validasi maupun testing.

Pembagian data kita lakukan sebagai berikut :

1. Data Training : 60%
2. Data Validasi : 20%
3. Data Testing : 20%

Lalu kita akan melakukan visualisasi data, dengan grafik, untuk data training kita beri warna titik biru, data test merah dan data validasi kuning.

```
[5] # Kita pecah 60% untuk training, 20 % untuk data test, dan sisanya (20%) untuk data validasi.
    TRAIN_SPLIT = int(0.6 * SAMPLES)
    TEST_SPLIT = int(0.2 * SAMPLES + TRAIN_SPLIT)


    # Kita gunakan np.split untuk membagi data menjadi 3 bagian (data train, data test, data validasi)
    x_train, x_test, x_validate = np.split(x_values, [TRAIN_SPLIT, TEST_SPLIT])
    y_train, y_test, y_validate = np.split(y_values, [TRAIN_SPLIT, TEST_SPLIT])

    # Cek apakah jumlah bagian (data train+data validasi+data test) sudah sama dengan total data (samples)
    assert (x_train.size + x_validate.size + x_test.size) == SAMPLES

    # Kita lakukan plot di grafik :
    plt.plot(x_train, y_train, 'b.', label="Train")
    plt.plot(x_test, y_test, 'r.', label="Test")
    plt.plot(x_validate, y_validate, 'y.', label="Validate")
    plt.legend()
    plt.show()
```

MELAKUKAN PEMECAHAN  
DATASET

Kita akan membuat model dengan beberapa lapis neuron.



```
#dibuat model berupa Sequential dari Keras, TensorFlow.
model = tf.keras.Sequential()

# Layer Pertama mengambil input tunggal dari data cos di atas, lalu diolah dalam 16 neuron, menggunakan 'relu' sebagai fungsi aktivasi.
model.add(keras.layers.Dense(16, activation='relu', input_shape=(1,)))

# Membuat layer ke dua dan ke tiga dengan 16 neuron untuk layer 2 dan 4 neuron untuk layer ketiga, masih menggunakan relu sebagai fungsi aktivasi.
model.add(keras.layers.Dense(16, activation='relu'))
model.add(keras.layers.Dense(4, activation='relu'))

# Output akhir berupa single neuron, yang melakukan output 1 variabel
model.add(keras.layers.Dense(1))

# Dilakukan pembentukan model dengan metode optimisasi:'adam'. Dilakukan evaluasi berupa Mean Squared Error (mse), yang merupakan loss dari metode regresi.
model.compile(optimizer='adam', loss="mse", metrics=["mae"])
```

## MELAKUKAN TRAINING MODEL

- Membuat Desain Model



# MELAKUKAN TRAINING MODEL

## ■ Training Pada Model

```
# Dilakukan training dengan epoch (siklus perulangan) sebanyak 500 kali. Melakukan training 64 data per epoch.
```

```
history = model.fit(x_train, y_train, epochs=500, batch_size=64,  
                    validation_data=(x_validate, y_validate))
```

```
# Jangan lupa model disimpan  
model.save(MODEL_TF)
```

```
Epoch 1/500  
5/5 [=====] - 1s 99ms/step - loss: 0.9109 - mae: 0.7664 - val_loss: 0.7110 - val_mae: 0.6814  
Epoch 2/500  
5/5 [=====] - 0s 9ms/step - loss: 0.6723 - mae: 0.6988 - val_loss: 0.6092 - val_mae: 0.6679  
Epoch 3/500  
5/5 [=====] - 0s 9ms/step - loss: 0.5523 - mae: 0.6555 - val_loss: 0.5749 - val_mae: 0.6741  
Epoch 4/500  
5/5 [=====] - 0s 10ms/step - loss: 0.5356 - mae: 0.6466 - val_loss: 0.5800 - val_mae: 0.6882  
Epoch 5/500  
5/5 [=====] - 0s 10ms/step - loss: 0.5019 - mae: 0.6254 - val_loss: 0.5945 - val_mae: 0.6971  
Epoch 6/500  
5/5 [=====] - 0s 10ms/step - loss: 0.5184 - mae: 0.6386 - val_loss: 0.6014 - val_mae: 0.7004  
Epoch 7/500  
5/5 [=====] - 0s 10ms/step - loss: 0.4984 - mae: 0.6234 - val_loss: 0.5990 - val_mae: 0.6989  
Epoch 8/500  
5/5 [=====] - 0s 10ms/step - loss: 0.4827 - mae: 0.6066 - val_loss: 0.5898 - val_mae: 0.6942  
Epoch 9/500  
5/5 [=====] - 0s 10ms/step - loss: 0.5242 - mae: 0.6409 - val_loss: 0.5798 - val_mae: 0.6881  
Epoch 10/500  
5/5 [=====] - 0s 10ms/step - loss: 0.5028 - mae: 0.6246 - val_loss: 0.5756 - val_mae: 0.6846  
Epoch 11/500  
5/5 [=====] - 0s 11ms/step - loss: 0.4989 - mae: 0.6194 - val_loss: 0.5736 - val_mae: 0.6825  
Epoch 12/500  
5/5 [=====] - 0s 11ms/step - loss: 0.5173 - mae: 0.6375 - val_loss: 0.5729 - val_mae: 0.6816  
Epoch 13/500  
5/5 [=====] - 0s 10ms/step - loss: 0.5037 - mae: 0.6262 - val_loss: 0.5741 - val_mae: 0.6831  
Epoch 14/500  
5/5 [=====] - 0s 9ms/step - loss: 0.5116 - mae: 0.6326 - val_loss: 0.5758 - val_mae: 0.6848  
Epoch 15/500  
5/5 [=====] - 0s 10ms/step - loss: 0.5270 - mae: 0.6442 - val_loss: 0.5770 - val_mae: 0.6858  
Epoch 16/500
```

# EVALUASI

- Plotting Parameter Output

```
# Dilakukan plotting hasil training tadi, menggunakan grafik loss (loss dan val_loss),

#1. Grafik Loss
train_loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(train_loss) + 1)
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.plot(epochs, train_loss, 'r.', label='Loss Saat Training')
plt.plot(epochs, val_loss, 'b.', label='Loss Saat Validasi')
plt.title('Grafik Loss Saat Training dan Validasi')
plt.xlabel('Epochs ke-')
plt.ylabel('Nilai Loss')
plt.legend()
plt.subplot(1, 2, 2)

# 2. Grafik mean absolute error
train_mae = history.history['mae']
val_mae = history.history['val_mae']
plt.plot(epochs, train_mae, 'g.', label='MAE saat Training')
plt.plot(epochs, val_mae, 'b.', label='MAE saat Validasi')
plt.title('Grafik Mean Absolute Error')
plt.xlabel('Epochs ke-')
plt.ylabel('MAE')
plt.legend()
plt.tight_layout()
```

```
# Kalkulasi hasil
test_loss, test_mae = model.evaluate(x_test, y_test)

# Membuat prediksi dengan dat test
y_test_pred = model.predict(x_test)

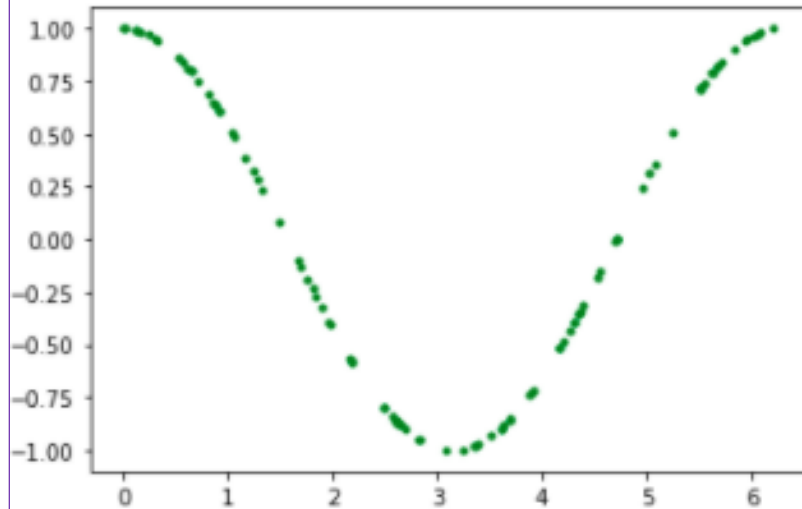
# Membuat grafik perbandingan nilai aktual dengan prediksi
plt.clf()
plt.title('Perbandingan grafik hasil prediksi dengan nilai aktual')
plt.plot(x_test, y_test, 'b.', label='Grafik Aktual')
plt.plot(x_test, y_test_pred, 'r.', label='Prediksi Model (TensorFlow)')
plt.legend()
plt.show()
```

## EVALUASI

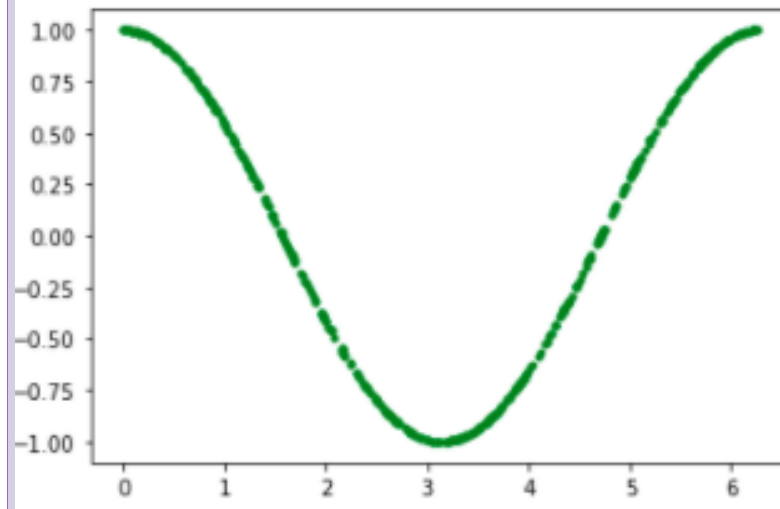
- Plotting Hasil Prediksi

## HASIL Pengerjaan : Data Grafik Cosinus

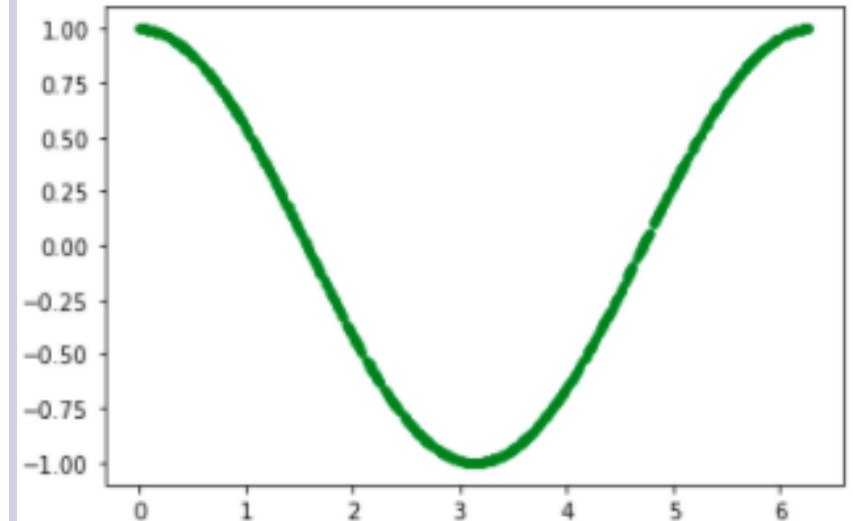
100 sampel



500 sampel

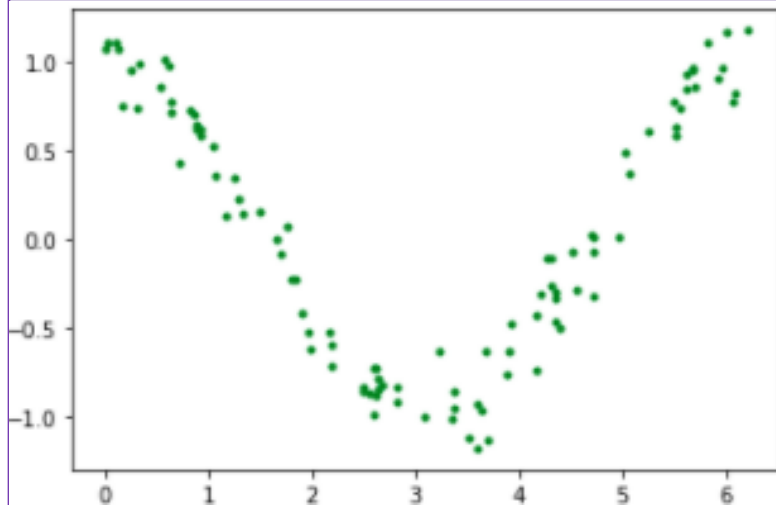


1000 sampel

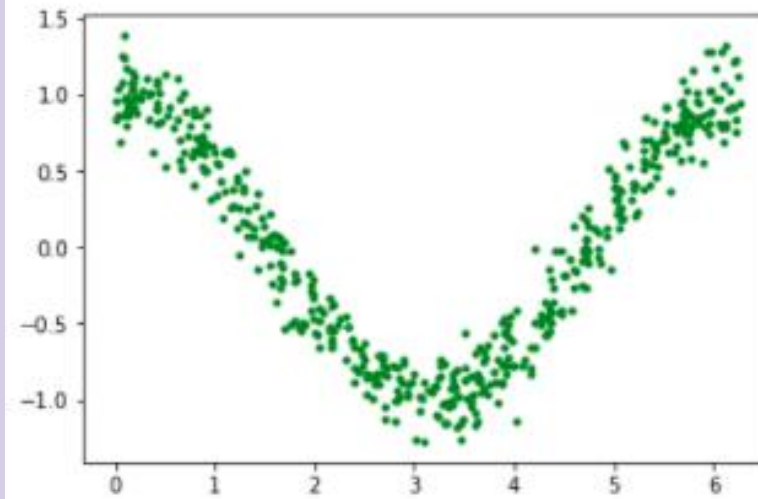


## HASIL Pengerjaan : Data Grafik Cosinus + Noise Random 15%

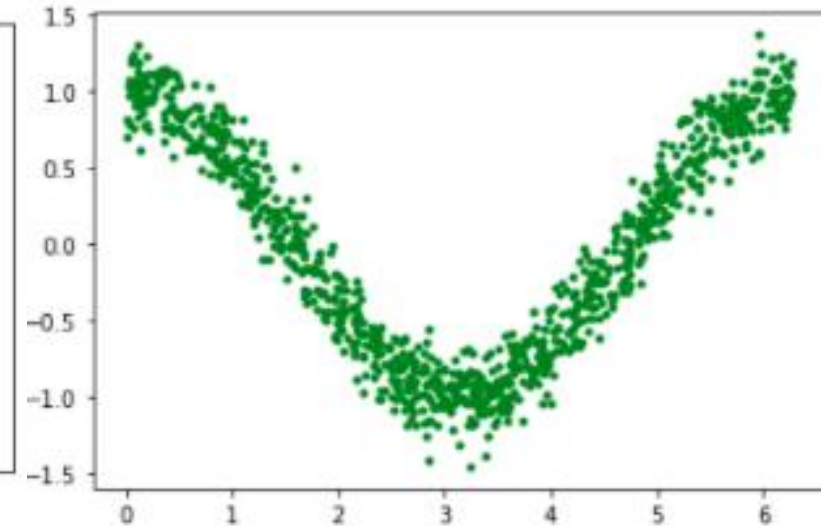
100 sampel



500 sampel

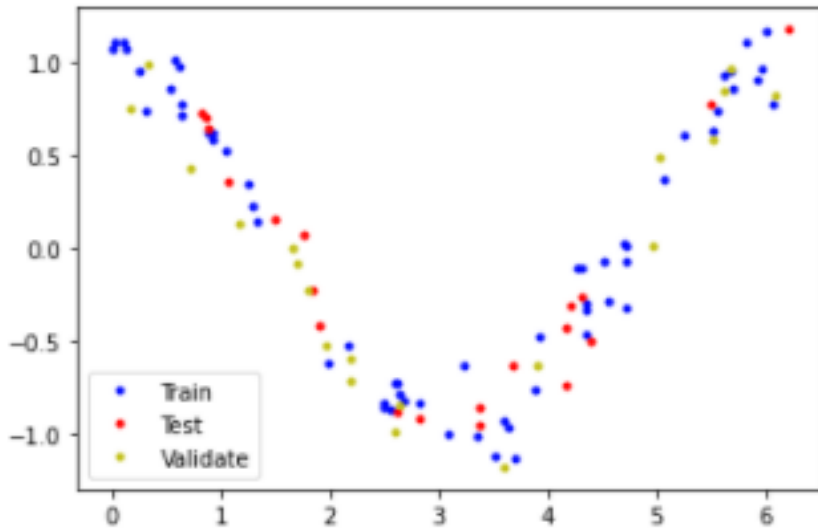


1000 sampel

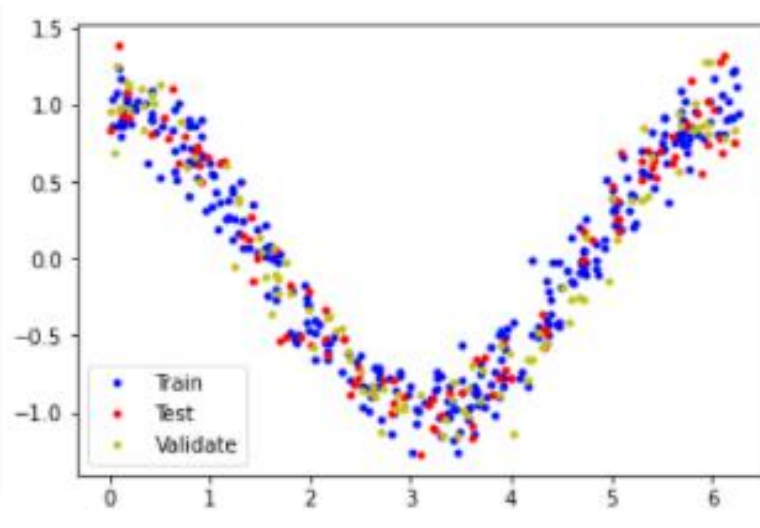


# HASIL PENGERJAAN : PEMECAHAN DATASET

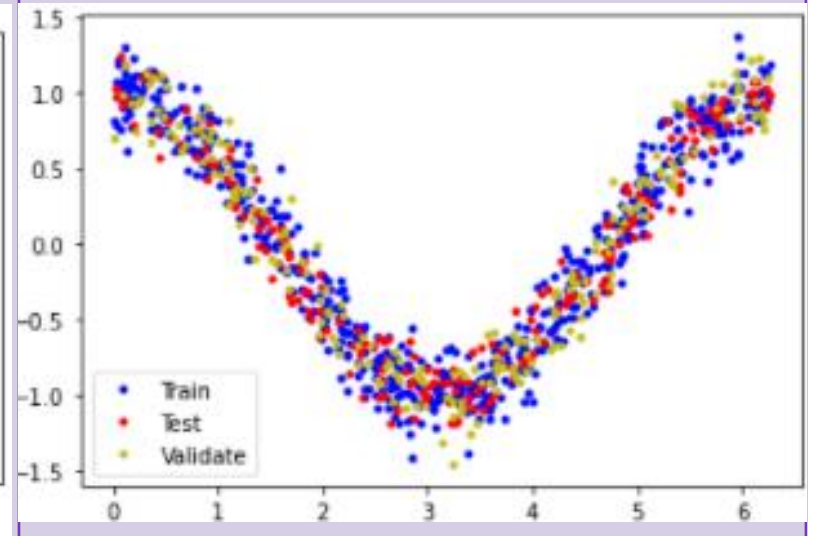
100 sampel



500 sampel



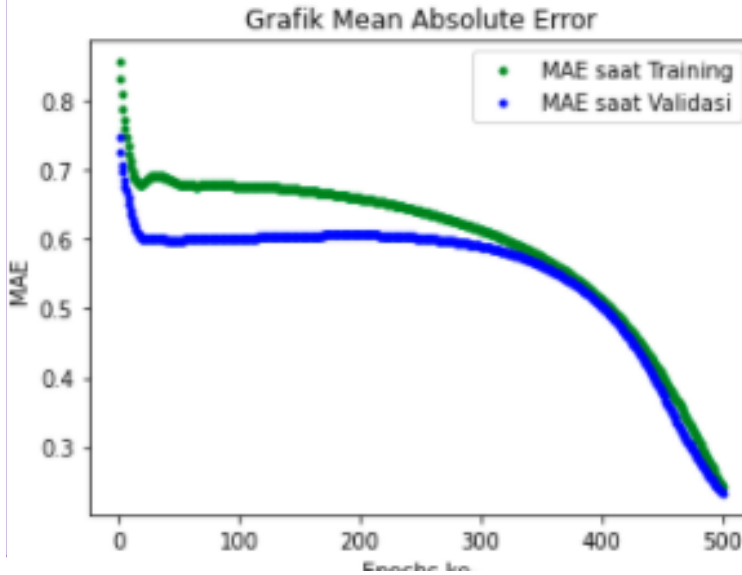
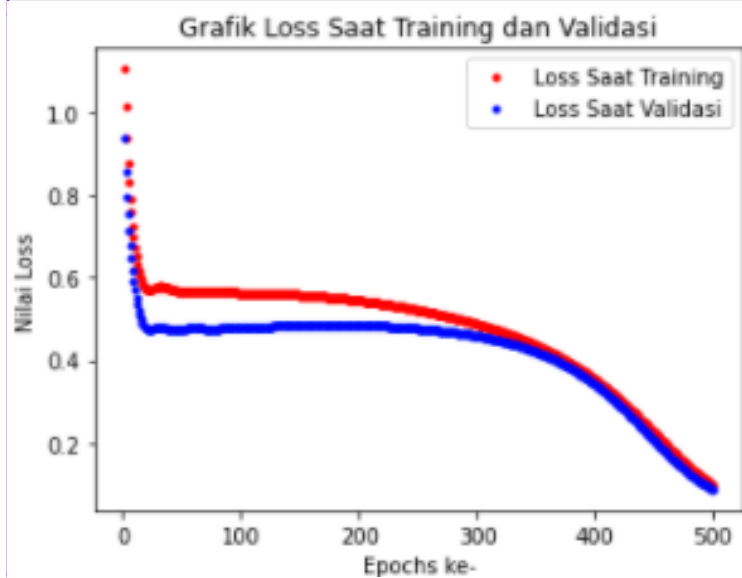
1000 sampel



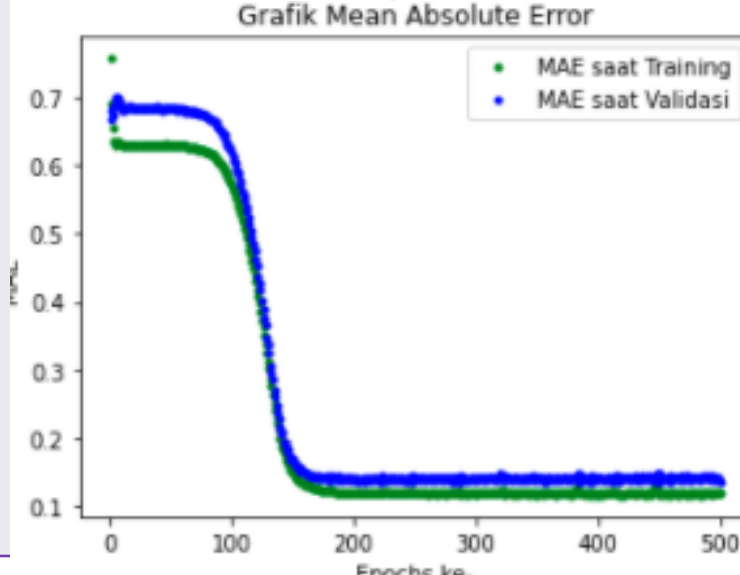
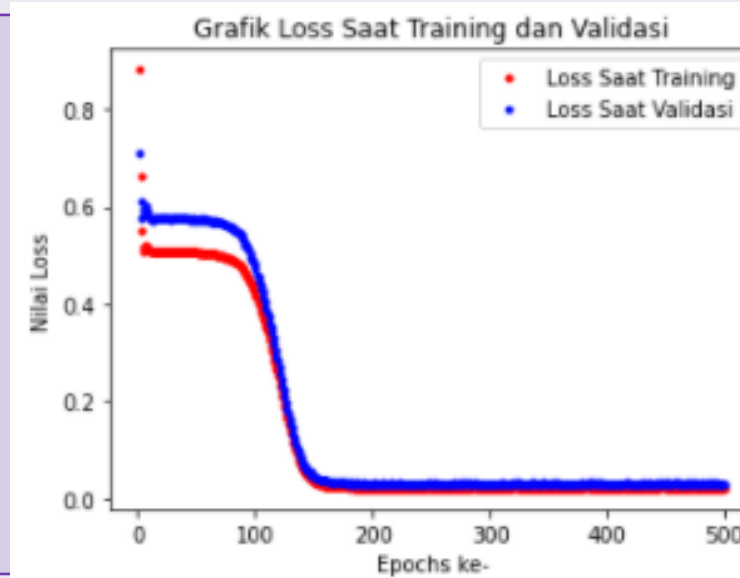


# HASIL Pengerjaan : Grafik Loss & MSE

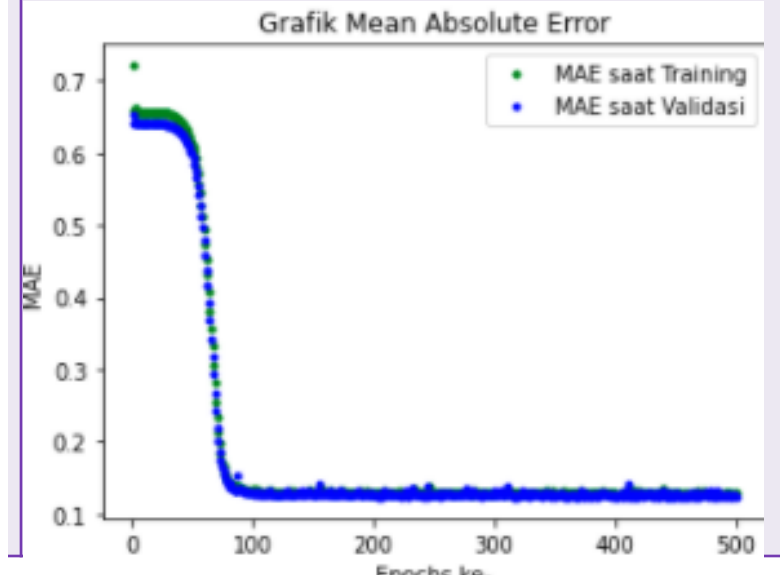
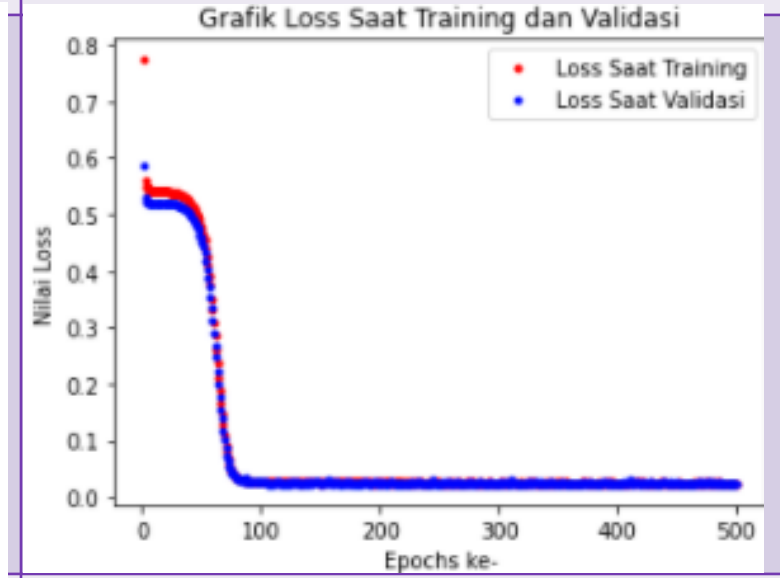
100 sampel



500 sampel



1000 sampel



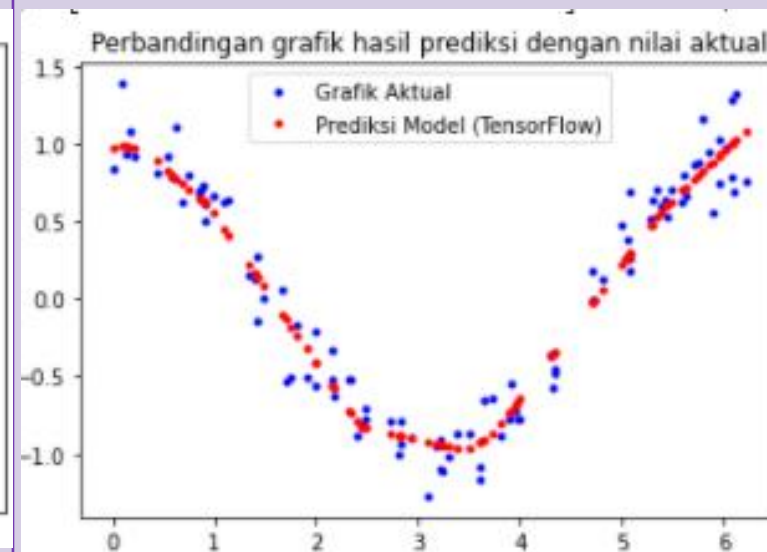
# HASIL Pengerjaan : Prediksi vs Aktual

100 sampel



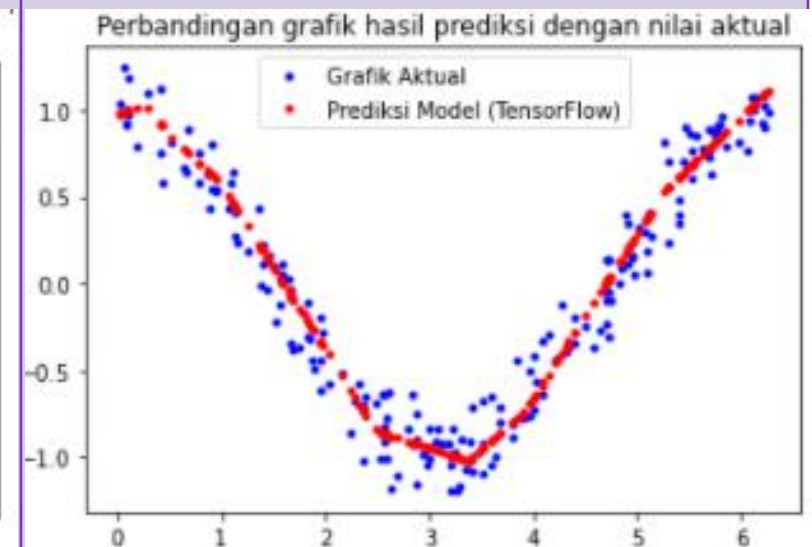
loss: 0.0953 - mae: 0.2608

500 sampel



loss: 0.0279 - mae: 0.1329

1000 sampel



loss: 0.0220 - mae: 0.1200

## ANALISIS DATA & KESIMPULAN

- Kenaikan jumlah sampel berkorelasi positif terhadap penurunan loss dan MSE.
- Pembuatan grafik dengan 100 sampel angka belum cukup merepresentasikan grafik fungsi cosinus, terutama setelah sampel dibagi menjadi data train, validasi, dan test.
- Dengan 500 sampel dapat dilihat bahwa loss dan MSE sudah cukup mendekati model dengan 1000 sampel.
- Semakin banyak sampel, maka jumlah epoch yang diperlukan semakin sedikit (grafik loss & MSE cukup rendah di epoch ke-500 pada penggunaan 100 sampel, cukup rendah pada epoch ke-150 pada penggunaan 500 sampel, dan cukup rendah pada epoch ke 70 pada penggunaan 1000 sampel).