VERSI 1.1

AGUSTUS 2025

PEMBELAJARAN MESIN

MODUL 4 - MENANGANI OVERFITTING DAN UNDERFITTING

DISUSUN OLEH:

YUFIS AZHAR S.KOM, M.KOM

KIARA AZZAHRA

ALVIYA LAELA LESTARI

TIM LABORATORIUM INFORMATIKA

UNIVERSITAS MUHAMMADIYAH MALANG

PENDAHULUAN

TUJUAN

1. Mahasiswa memahami konsep overfitting dan underfitting dalam pembelajaran mesin.
2. Mahasiswa mampu mengidentifikasi penyebab utama overfitting dan underfitting pada model machine learning.
3. Mahasiswa memahami berbagai metode untuk mengatasi overfitting, termasuk regularization, dropout, early stopping, fine-tuning, dan data augmentation.
4. Mahasiswa memahami metode untuk mengatasi underfitting, termasuk peningkatan kompleksitas model, feature engineering, dan memperpanjang durasi training.
5. Mahasiswa mampu membedakan strategi penanganan berdasarkan jenis data (tabular, teks, dan citra).
6. Mahasiswa mampu mengimplementasikan dan mengevaluasi strategi penanganan overfitting dan underfitting menggunakan Python, TensorFlow, dan Keras.

TARGET MODUL

1. Mahasiswa mampu mengenali tanda-tanda model mengalami overfitting maupun underfitting melalui hasil evaluasi model.
2. Mahasiswa mampu menerapkan preprocessing, tuning model, serta augmentasi data untuk meningkatkan performa model.
3. Mahasiswa mampu melakukan eksperimen pada berbagai jenis data (tabular, teks, citra) untuk mengatasi permasalahan fitting.
4. Mahasiswa mampu menghasilkan model yang lebih **general** (tidak overfit) dan **powerful** (tidak underfit) dengan akurasi yang lebih stabil pada data validasi maupun data uji.

PERSIAPAN

1. Laptop/PC
2. Interactive Phyton Notebook (ipynb)
3. Python (direkomendasikan menggunakan versi >= 3.10
4. TensorFlow (direkomendasikan menggunakan versi >= 2.10)
5. Keras, Matplotlib, seaborn, scikit-learn, pandas, numpy.

KEYWORDS

Overfitting, Underfitting, Regularization, Dropout, Early Stopping, Fine Tuning, Data Augmentation, Feature Engineering, CNN, Deep Learning

TABLE OF CONTENTS

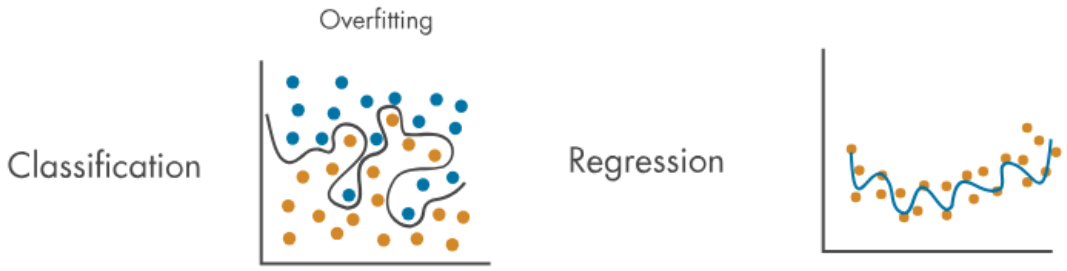
MATERI POKOK

Google Colab Modul 4: LINK COLAB

A. Overfitting

Overfitting adalah kejadian ketika model *machine learning* dilatih terlalu baik pada data *train* sehingga model menjadi terlalu menghafal pola, bahkan noise atau detail acak yang tidak relevan. Akibatnya, model kehilangan kemampuan untuk menggeneralisasi atau membuat prediksi yang akurat pada data baru yang belum pernah dilihat sebelumnya (data *valid* atau data *test*). Ibaratnya, seperti seorang mahasiswa yang menghafal seluruh contoh soal dan jawaban di buku, tanpa benar-benar memahami konsepnya. Jadi, ketika diberikan soal baru yang sedikit berbeda, mahasiswa tersebut akan kesulitan menjawab. Sama halnya dengan *overfitting*, model

akan berkinerja sangat baik pada data yang sudah dikenali (data *train*), tetapi gagal saat diberikan data baru yang belum pernah dilihat sebelumnya.



Grafik *Overfitting* (source: mathworks.com)

Dari grafik diatas adalah contoh grafik overfitting, titik-titik biru dan oranye mewakili dua kategori data yang berbeda. Garis hitam yang berliku-liku adalah *decision boundary* yang dibuat oleh model. Perhatikan garis hitam pada grafik *classification* yang berliku-liku untuk mengurung setiap titik biru dan oranye, bahkan sampai ada titik biru yang dikelilingi sepenuhnya. Ini menunjukkan bahwa model terlalu fokus pada data *train* dan tidak akan bisa menggeneralisasi dengan baik jika ada titik data baru yang letaknya sedikit bergeser. Sedangkan pada grafik *regression* dapat dilihat bahwa model membuat garis prediksi yang sangat tidak mulus dan berbelok-belok tajam untuk "mencocokkan" setiap titik data *train* dengan sempurna.

Ciri-ciri utama model mengalami overfitting biasanya dapat dilihat dari performa yang tinggi pada data *train* dibandingkan dengan akurasi data *valid.* Akurasi data *train* bahkan bisa mencapai 99% atau 100%, sedangkan akurasi performa pada data *valid/test* sangat berbanding jauh lebih rendah dengan data *train*.

Berikut beberapa penyebab *overfitting*:

1. Model terlalu kompleks, terlalu banyak parameter atau lapisan (misalnya terlalu banyak hidden layer) yang membuat model terlalu banyak "kebebasan" untuk menghafal data.
2. Data *train* terlalu sedikit sehingga tidak cukup untuk mewakili seluruh pola yang mungkin ada. Akibatnya, model hanya belajar dari sedikit contoh yang tersedia.
3. Durasi *training* terlalu lama dimana model dilatih hingga mencapai kondisi optimal pada data *train,* melampaui titik dimana model mulai kehilangan kemampuan generalisasi.

Untuk mengatasi *overfitting* kita bisa mencoba beberapa teknik umum untuk mengatasinya:

1. **Validasi Silang (Cross-Validation)**

Data dibagi menjadi beberapa subset. Model kemudian dilatih dan divalidasi secara bergantian menggunakan subset yang berbeda. Dengan cara ini, model benar-benar teruji pada data yang beragam sehingga hasil evaluasi lebih dapat dipercaya.

2. **Regularisasi (Regularization)**

Memberikan penalti pada model jika terlalu kompleks. Tujuannya agar model tidak sekadar "menghafal" data. Dua teknik yang sering digunakan adalah L1 (Lasso) dan L2 (Ridge) *regularization*.

3. ***Dropout***

Pada saat pelatihan, beberapa neuron dipilih secara acak untuk "dimatikan" sementara. Hal ini mencegah model terlalu bergantung pada neuron tertentu dan membantu meningkatkan kemampuan generalisasi.

4. ***Early Stopping***

Proses pelatihan dipantau melalui *validation loss*. Jika validation loss mulai meningkat (tanda model mulai *overfitting*), maka proses pelatihan dihentikan secara otomatis.

5. **Augmentasi Data (Data Augmentation)**

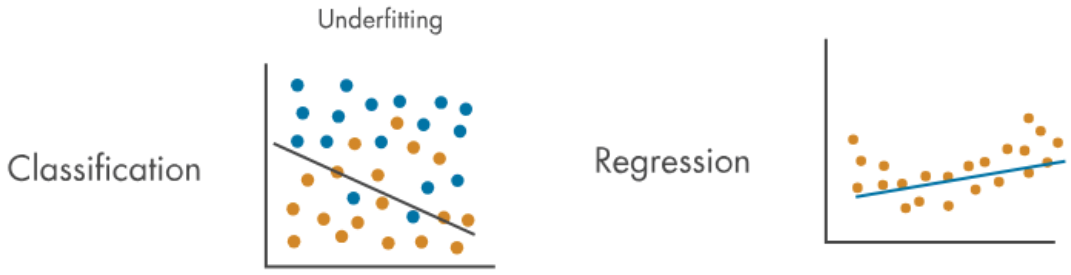
Teknik ini memperbanyak variasi data latih dengan memodifikasi data yang sudah ada, seperti memutar, membalik, atau mengubah skala gambar. Tujuannya agar model belajar dari data yang lebih beragam.

6. **Seleksi Fitur (Feature Selection)**

Proses memilih fitur-fitur yang benar-benar relevan dan membuang fitur yang tidak penting. Dengan begitu, kompleksitas berkurang, *noise* diminimalkan, dan model menjadi lebih efisien.

B. Underfitting

Underfitting adalah kebalikan dari *overfitting,* yaitu masalah dalam *machine learning* dimana model yang kita bangun terlalu sederhana untuk menangkap pola-pola dasar dari data. Akibatnya, model memiliki kinerja yang buruk, baik pada data *train* maupun data baru. Jika overfitting diibaratkan sebagai mahasiswa yang menghafal terlalu banyak detail, maka *underfitting* adalah seperti mahasiswa yang tidak belajar sama sekali. Model tidak memahami konsep dasar data, sehingga gagal dalam tugasnya, baik untuk data yang sudah dilihat maupun yang belum.



Grafik *underfitting* (Source: mathworks.com)

Grafik diatas ini adalah contoh *underfitting*. Titik-titik biru dan oranye mewakili dua kategori data yang berbeda. Garis lurus hitam adalah *decision boundary* yang dibuat oleh model. Perhatikan garis lurus pada grafik *classification* yang tidak mampu memisahkan titik-titik biru

dan oranye dengan baik. Ini menunjukkan bahwa model terlalu sederhana dan tidak bisa menangkap pola data, sehingga banyak titik yang salah diklasifikasikan.

Sedangkan pada grafik *regression*, dapat dilihat bahwa model membuat garis prediksi yang sangat tidak akurat, yaitu garis lurus. Model gagal mengikuti pola data yang berliku, karena garis prediksinya tidak mampu melewati sebagian besar titik data.

Ciri-ciri utama model mengalami *underfitting* biasanya dapat dilihat dari performa yang **rendah** pada data *train* maupun data *valid*. Model tidak bisa berkinerja baik bahkan pada data yang digunakan untuk melatihnya.

Berikut beberapa penyebab *underfitting*:

1. Model terlalu sederhana, tidak memiliki cukup parameter atau lapisan yang membuatnya tidak bisa menangkap kompleksitas data.
2. Data *train* terlalu sedikit atau tidak memiliki fitur yang relevan untuk membantu model membuat keputusan yang akurat.
3. Durasi *training* terlalu singkat, di mana model belum dilatih cukup lama untuk belajar dari data yang tersedia.

Untuk mengatasi *overfitting* kita bisa mencoba beberapa teknik umum untuk mengatasinya:

1. **Menambah Kompleksitas Model**

Underfitting biasanya terjadi karena model terlalu sederhana. Solusinya adalah menggunakan arsitektur model yang lebih kompleks atau menambah jumlah lapisan/neuron agar model mampu menangkap pola data dengan lebih baik.

2. **Mengurangi Regularisasi**

Regularisasi yang terlalu kuat bisa membuat model kesulitan belajar pola. Dengan mengurangi nilai regularisasi (misalnya parameter lambda pada L1/L2), model diberi ruang untuk belajar lebih fleksibel.

3. ***Training* Lebih Lama**

Underfitting dapat muncul jika pelatihan dihentikan terlalu cepat. Dengan menambah jumlah epoch, model punya kesempatan lebih banyak untuk mempelajari pola dari data.

4. ***Feature Engineering* yang Lebih Baik**

Terkadang model *underfit* karena fitur yang digunakan kurang informatif. Dengan menambahkan fitur baru, menggabungkan, atau mengubah representasi fitur, model dapat memahami data lebih baik.

5. **Mengurangi Noise pada Data**

Data yang terlalu bising membuat pola penting sulit dipelajari. Membersihkan data atau melakukan *preprocessing* yang tepat dapat membantu model belajar lebih akurat.

6. **Menggunakan Data yang Lebih Relevan**

Jika dataset terlalu kecil atau tidak representatif, model akan kesulitan mengenali pola. Menambahkan data baru yang lebih berkualitas atau relevan dapat mengurangi risiko *underfitting*.

C. Metode Untuk Mengatasi Overfitting dan Underfitting

1. Mengatasi Underfitting

Untuk mengatasi masalah *underfitting,* metode yang paling oke adalah mengganti model yang terlalu sederhana menjadi lebih kompleks dan menyesuaikan dengan kebutuhan data yang kita punya. Berikut beberapa contoh mengatasi masalah *underfitting* pada jenis data yang berbeda.

a. Data Tabular

Pertama, kita akan membuat data tabular secara manual menggunakan numpy.

```
# Buat data tabular sederhana
np.random.seed(0)
X = np.sort(5 * np.random.rand(80, 1), axis=0)
y = np.sin(X).ravel() + np.random.normal(0, 0.1, X.shape[0])
```

Kemudian, lakukan splitting data

```
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Setelah data sudah siap untuk diolah, mari kita lakukan percobaan dengan model yang sangat sederhana.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
# Simple Neural Network (Underfitting)
simple_nn_model = Sequential([
    Dense(4, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(1)
])

simple_nn_model.compile(optimizer='adam', loss='mse')
simple_nn_history = simple_nn_model.fit(X_train, y_train, epochs=50,
batch_size=32, verbose=0)

print("Underfitting dengan Neural Network Sederhana:")
print(f"MSE Pelatihan: {simple_nn_model.evaluate(X_train, y_train,
verbose=0):.4f}")
print(f"MSE Pengujian: {simple_nn_model.evaluate(X_test, y_test,
verbose=0):.4f}\n")
```

Model diatas akan menghasilkan nilai MSE sebagai berikut:

```
Underfitting dengan Neural Network Sederhana:
MSE Pelatihan: 0.3582
MSE Pengujian: 0.5042
```

Untuk mengatasi permasalahan underfitting, maka kita coba buat model yang lebih kompleks:

```
# More Complex Neural Network (Addressing Underfitting)
complex_nn_model = Sequential([
    Dense(32, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(1)
])

complex_nn_model.compile(optimizer='adam', loss='mse')
complex_nn_history = complex_nn_model.fit(X_train, y_train, epochs=50,
batch_size=32, verbose=0)

print("Mengatasi Underfitting dengan Neural Network yang Lebih Kompleks:")
print(f"MSE Pelatihan: {complex_nn_model.evaluate(X_train, y_train,
verbose=0):.4f}")
print(f"MSE Pengujian: {complex_nn_model.evaluate(X_test, y_test,
verbose=0):.4f}\n")
```

Setelah dilakukan training, model akan menghasilkan nilai MSE sebagai berikut:

```
Mengatasi Underfitting dengan Neural Network yang Lebih Kompleks:
MSE Pelatihan: 0.1973
MSE Pengujian: 0.3143
```

Dengan menambahkan neuron dan beberapa lapisan yang lebih kompleks dibandingkan model sebelumnya, nilai MSE jauh lebih rendah dibandingkan MSE dengan model yang kurang kompleks. Mari kita coba dengan menambahkan epochsnya.

```
# More Complex Neural Network (Addressing Underfitting)
complex_nn_model = Sequential([
    Dense(32, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(1)
])

complex_nn_model.compile(optimizer='adam', loss='mse')
```

```
complex_nn_history = complex_nn_model.fit(X_train, y_train, epochs=100,
batch_size=32, verbose=0) #menambahkan epochs

print("Mengatasi Underfitting dengan Neural Network yang Lebih Kompleks:")
print(f"MSE Pelatihan: {complex_nn_model.evaluate(X_train, y_train,
verbose=0):.4f}")
print(f"MSE Pengujian: {complex_nn_model.evaluate(X_test, y_test,
verbose=0):.4f}\n")
```

Hasil kode di atas akan menghasilkan nilai MSE berikut:

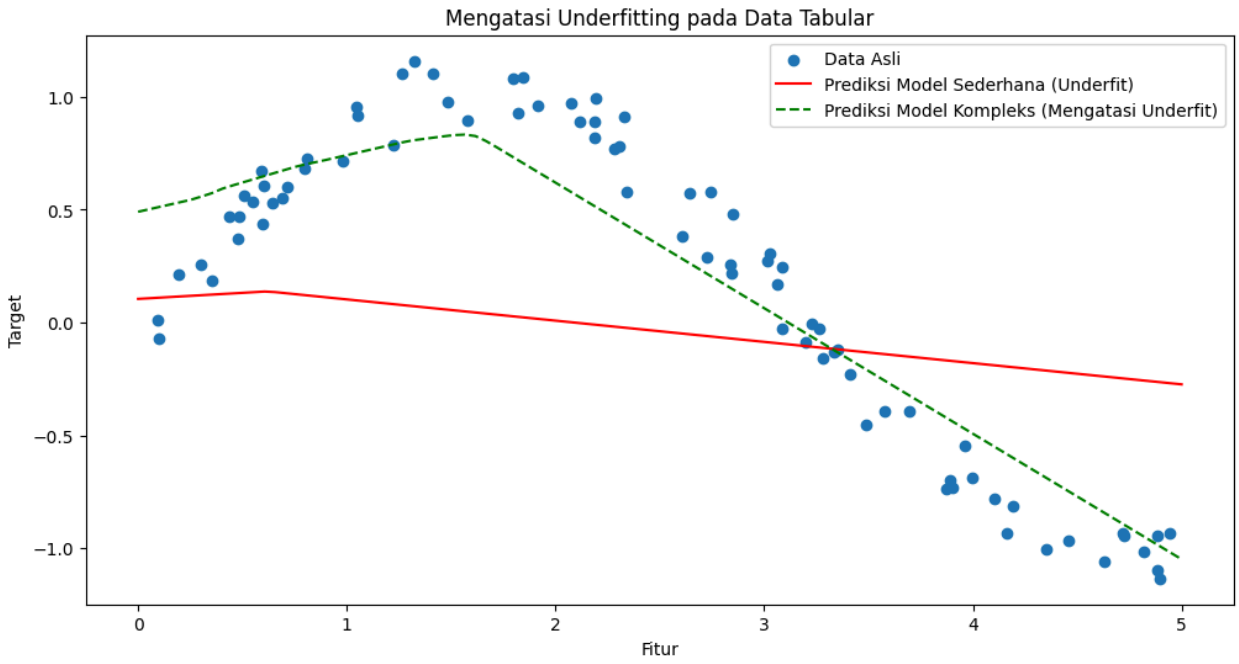
```
Mengatasi Underfitting dengan Neural Network yang Lebih Kompleks:
MSE Pelatihan: 0.0482
MSE Pengujian: 0.0904
```

Nilai MSE jauh lebih rendah lagi jika kita menambahkan epochs model untuk diberikan training. Namun, perlu di notes, bukan berarti semakin kompleks suatu model maka akan semakin baik akurasinya. Karena, jika terlalu berlebihan, maka kita akan menemui masalah lain yaitu overfitting. Sehingga, kita harus melakukan tuning parameter model hingga mendapat performa terbaiknya.

Gunakan kode berikut untuk mengetahui visualisasi dari hasil training model

```
X_plot = np.linspace(0, 5, 100)[:, np.newaxis]
y_plot_simple = simple_nn_model.predict(X_plot)
y_plot_complex = complex_nn_model.predict(X_plot)

plt.figure(figsize=(12, 6))
plt.scatter(X, y, label='Data Asli')
plt.plot(X_plot, y_plot_simple, color='red', label='Prediksi Model Sederhana
(Underfit)')
plt.plot(X_plot, y_plot_complex, color='green', linestyle='--', label='Prediksi
Model Kompleks (Mengatasi Underfit)')
plt.title('Mengatasi Underfitting pada Data Tabular')
plt.xlabel('Fitur')
plt.ylabel('Target')
plt.legend()
plt.show()
```

Hasil kode di atas, akan menghasilkan visualisasi berikut:



b. Data Teks/Citra

Pertama, kita akan menggunakan data MNIST yang sudah disediakan oleh Keras:

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, MaxPooling2D
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# Muat data MNIST
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

Selanjutnya, lakukan preprocessing data terlebih dahulu

```
# Pre-processing data
X_train = X_train.reshape((60000, 28, 28, 1)).astype('float32') / 255
X_test = X_test.reshape((10000, 28, 28, 1)).astype('float32') / 255
y_train_cat = to_categorical(y_train)
y_test_cat = to_categorical(y_test)
```

Kemudian, buat model sederhananya.

```
simple_model_cnn = Sequential([
    Conv2D(16, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    Flatten(),
    Dense(10, activation='softmax')
```

```
])

simple_model_cnn.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
simple_history = simple_model_cnn.fit(X_train, y_train_cat, epochs=1,
batch_size=64, validation_split=0.2)
print("\nUnderfitting dengan model sederhana (1 epoch):")
print(f"Akurasi pelatihan: {simple_history.history['accuracy'][-1]:.4f}")
print(f"Akurasi validasi: {simple_history.history['val_accuracy'][-1]:.4f}\n")
```

Hasil dari kode diatas akan menghasilkan akurasi sebagai berikut:

```
Underfitting dengan model sederhana (1 epoch):
Akurasi pelatihan: 0.9280
Akurasi validasi: 0.9697
```

Walaupun akurasi validasi terlihat cukup baik, akurasi pada data latih yang masih rendah dan durasi pelatihan yang singkat menunjukkan bahwa model belum benar-benar mampu menangkap pola dari data latih. Jika pelatihan pada model sederhana ini dilanjutkan, kemungkinan akurasi latih akan sedikit membaik, namun akan cepat mencapai titik jenuh yang kurang optimal. Mari kita buat model yang lebih kompleks:

```
      complex_model_cnn = Sequential([
          Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
          MaxPooling2D((2, 2)),
          Conv2D(64, (3, 3), activation='relu'),
          MaxPooling2D((2, 2)),
          Flatten(),
          Dense(128, activation='relu'),
          Dense(10, activation='softmax')
      ])

      complex_model_cnn.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
      complex_history = complex_model_cnn.fit(X_train, y_train_cat, epochs=1,
batch_size=64, validation_split=0.2)
      print("Mengatasi Underfitting dengan arsitektur yang lebih kompleks:")
      print(f"Akurasi pelatihan:
{complex_history.history['accuracy'][-1]:.4f}")
```

```
    print(f"Akurasi validasi:
{complex_history.history['val_accuracy'][-1]:.4f}\n")
```

Hasilnya adalah:

```
750/750 ━━━━━━━━━━━━━━━━━━━━ 42s 53ms/step - accuracy: 0.8803 - loss: 0.4067 - val_accuracy: 0.9762 - val_loss: 0.0776
Mengatasi Underfitting dengan arsitektur yang lebih kompleks:
Akurasi pelatihan: 0.9477
Akurasi validasi: 0.9762
```

Bandingkan dengan akurasi diatas ini, dengan menggunakan model yang lebih kompleks, bahkan setelah hanya satu epoch, sudah mencapai akurasi pelatihan yang lebih tinggi (0.9477) dan akurasi validasi yang juga lebih tinggi (0.9762) dibandingkan model sederhana. Ini menunjukkan bahwa arsitektur yang lebih kompleks lebih cepat dalam mempelajari pola dalam data. Mari kita coba training ulang model yang kurang kompleks dengan menambahkan jumlah epochsnya.

```
long_training_history = complex_model_cnn.fit(X_train, y_train_cat, epochs=5,
batch_size=64, validation_split=0.2)
print("Mengatasi Underfitting dengan melatih lebih lama (5 epochs):")
print(f"Akurasi pelatihan:
{long_training_history.history['accuracy'][-1]:.4f}")
print(f"Akurasi validasi:
{long_training_history.history['val_accuracy'][-1]:.4f}\n")
```

Hasilnya adalah:

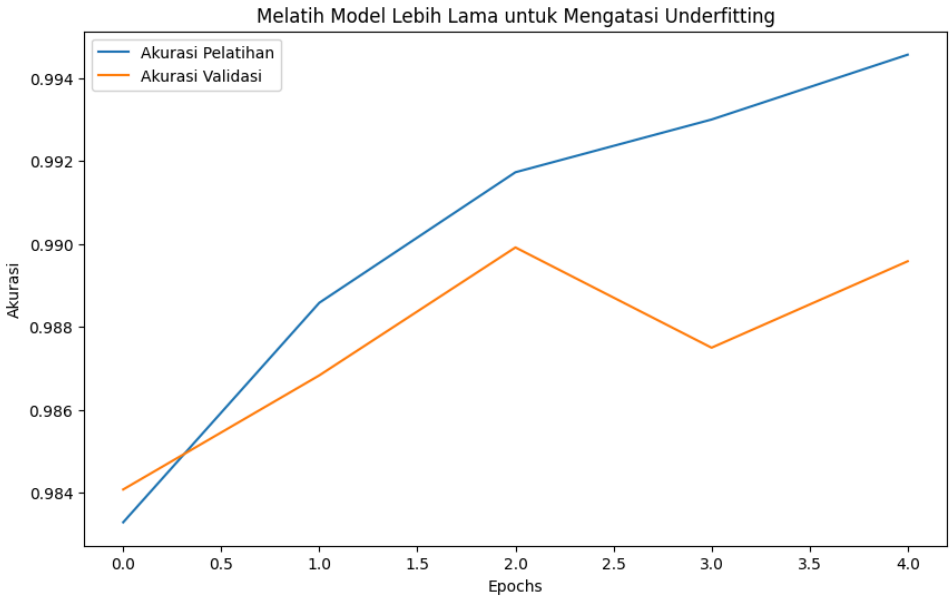
```
Epoch 5/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 39s 52ms/step - accuracy: 0.9950 - loss: 0.0151 - val_accuracy: 0.9896 - val_loss: 0.0379
Mengatasi Underfitting dengan melatih lebih lama (5 epochs):
Akurasi pelatihan: 0.9946
Akurasi validasi: 0.9896
```

Akurasi pelatihan naik tajam hingga 0.9946 dan akurasi validasi juga mencapai 0.9896. Hal ini menunjukkan bahwa dalam satu epoch, baik model sederhana maupun kompleks masih memiliki ruang yang besar untuk belajar, sehingga model belum sepenuhnya memanfaatkan informasi dari data latih. Kondisi ini menjadi tanda adanya underfitting.

Singkatnya, pada CNN sederhana dengan pelatihan hanya 1 epoch, *underfitting* terlihat dari performa pelatihan yang masih bisa ditingkatkan secara signifikan melalui penambahan kompleksitas model maupun durasi pelatihan. Meskipun akurasi validasi di akhir epoch pertama lebih tinggi daripada akurasi latih, hal ini tidak berarti model sudah optimal, karena sebenarnya model belum benar-benar “belajar” secara menyeluruh.

```
# Plot akurasi seiring epochs untuk melihat underfitting
plt.figure(figsize=(10, 6))
plt.plot(long_training_history.history['accuracy'], label='Akurasi Pelatihan')
```

```
plt.plot(long_training_history.history['val_accuracy'], label='Akurasi
Validasi')
plt.title('Melatih Model Lebih Lama untuk Mengatasi Underfitting')
plt.xlabel('Epochs')
plt.ylabel('Akurasi')
plt.legend()
plt.show()
```



2. Mengatasi Overfitting

a. Overfitting Data Teks dan Data Citra Menggunakan Metode Paling Umum

Metode-metode ini adalah fondasi dalam mencegah overfitting dan bisa diterapkan pada berbagai jenis neural network, termasuk yang digunakan untuk data teks dan citra.

REGULARIZATION

Bayangkan kita sedang mengajari model untuk "menghitung" seberapa penting setiap fitur. Regularisasi adalah teknik untuk "menghukum" model jika ia memberikan bobot yang terlalu besar pada fitur tertentu. Dengan bobot yang lebih kecil dan merata, model menjadi lebih sederhana dan tidak mudah overfit.

- **L1 (Lasso)**: menambah penalti absolute value bobot → membuat model lebih sederhana dengan nge-"nol"-in koefisien yang nggak penting.
- **L2 (Ridge)**: menambah penalti kuadrat bobot → membuat bobot nggak terlalu besar, jadi lebih stabil. L2 Regularization lebih sering dipakai karena ia cenderung membuat bobot-bobot menjadi lebih kecil secara umum, yang sangat membantu menstabilkan model kita.

```
import tensorflow as tf
```

```
from tensorflow.keras import layers, regularizers

# Contoh: Model dengan L2 Regularization
model = tf.keras.Sequential([
    layers.Dense(64, activation='relu',
kernel_regularizer=regularizers.l2(0.001), input_shape=(...)),
    layers.Dense(64, activation='relu',
kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(1, activation='sigmoid')
])
```

DROPOUT

Dropout adalah salah satu teknik paling efektif dan mudah dipakai. Pada setiap iterasi pelatihan, dropout akan secara acak "mematikan" beberapa neuron. Hal ini mencegah neuron-neuron tersebut untuk saling "kompak" atau terlalu bergantung pada satu sama lain. Hasilnya, jaringan dipaksa untuk belajar fitur yang lebih beragam dan umum, tidak hanya mengandalkan beberapa neuron saja.

```
import tensorflow as tf
from tensorflow.keras import layers

# Contoh: Model dengan Dropout
model = tf.keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=(...)),
    layers.Dropout(0.5), # 50% neuron akan 'dimatikan'
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(1, activation='sigmoid')
])
```

EARLY STOPPING

Seperti yang sudah sempat di mention di modul 2, daripada melatih model sampai selesai jumlah epoch yang kita tentukan, early stopping akan memantau performa model pada data validasi. Jika performanya tidak membaik selama beberapa epoch (misalnya, selama 5 epoch berturut-turut), pelatihan akan dihentikan. Ini mencegah model untuk terus belajar noise dari data latih yang bisa menyebabkan *overfitting*. Sebenarnya masih ada beberapa jenis callback dari Keras, cek di link berikut: *[LINK API CALLBACK KERAS]*

```
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.callbacks import EarlyStopping

# Definisikan callback
early_stopping = EarlyStopping(
monitor='val_loss', # Pantau loss pada data validasi
```

```
patience=5,          # Hentikan jika tidak ada perbaikan selama 5 epoch
restore_best_weights=True # Kembalikan bobot terbaik
      )
model = tf.keras.Sequential([
          layers.Dense(64, activation='relu', input_shape=(...)),
          layers.Dense(1, activation='sigmoid')
      ])

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

      # Gunakan callback saat melatih
history = model.fit(
       X_train, y_train,
       epochs=100,
       validation_data=(X_val, y_val),
       callbacks=[early_stopping] # Panggil early_stopping disini
      )
```

Selain metode di atas, masih ada beberapa metode khusus yang dapat digunakan untuk menangani jenis data yang berbeda. Baca, pahami, dan praktekkan materi berikut ini.

FINE TUNING MODEL

Dalam proses fine-tuning model *machine learning*, kita biasanya melakukan perubahan secara bertahap pada parameter-parameter model untuk mencari kombinasi terbaik. Setiap perubahan yang kita lakukan sebaiknya dicatat hasilnya, agar kita tahu arah perbaikan dan tidak mengulang kesalahan yang sama.

Sebagai contoh, pada percobaan pertama mungkin model kita mengalami underfitting (akurasi rendah, baik di data latih maupun validasi). Dari sini, kita bisa menganalisis kemungkinan penyebabnya: misalnya arsitektur model terlalu sederhana. Solusi yang dapat dicoba adalah menambahkan *hidden layer* atau menambah jumlah neuron pada layer yang sudah ada. Jika setelah ditambahkan ternyata akurasi meningkat, tetapi masih belum mencapai target, kita lanjutkan ke percobaan berikutnya.

Namun, pada percobaan kedua misalnya kita menambahkan *hidden layer* lagi dan hasilnya justru menurun, berarti menambah layer terus-menerus bukan solusi. Dalam kondisi seperti ini, kita bisa mencoba parameter lain, misalnya:

- Menambah jumlah neuron pada layer tertentu.
- Mengganti fungsi aktivasi agar model mampu menangkap non-linearitas lebih baik.
- Mengubah *optimizer* (misalnya dari SGD ke Adam, atau menyesuaikan learning rate).
- Menambahkan *regularizer* (seperti *Dropout* atau *L2 Regularization*) untuk mencegah *overfitting*.

- Melakukan augmentasi data (khusus untuk data citra/teks) agar model lebih general.
- Menyesuaikan *batch size* dan jumlah *epoch* saat *training*.

Dengan cara ini, proses tuning lebih terstruktur karena setiap eksperimen punya catatan, kita tahu perubahan apa yang dilakukan, hasilnya bagaimana, dan langkah apa yang selanjutnya bisa diambil.

Contoh Log Fine-Tuning Model

No	Parameter yang Diubah	Nilai/Setting	Hasil Training (Akurasi/Loss)	Catatan Analisis	Keputusan Lanjut
1	Arsitektur (hidden layer)	1 hidden layer (64 neuron)	Train acc: 65% Val acc: 62%	Model **underfitting**, terlalu sederhana	Tambah hidden layer
2	Arsitektur (hidden layer)	2 hidden layer (64 neuron)	Train acc: 78% Val acc: 74%	Akurasi meningkat, tapi masih rendah	Tambah neuron
3	Jumlah neuron	2 hidden layer (128 neuron)	Train acc: 88% Val acc: 80%	Gap train > val → potensi overfitting	Coba regularizer
4	Regularizer (Dropout)	Dropout 0.3 di layer 1	Train acc: 84% Val acc: 82%	Generalisasi membaik, gap menurun	Stabilkan LR
5	Optimizer & Learning Rate	Adam, lr=0.0005	Train acc: 87% Val acc: 86%	Model stabil, akurasi mendekati target	Stop (cukup)

b. Overfitting Pada Data Citra

AUGMENTASI DATA

Augmentasi data adalah trik cerdas untuk memperbanyak data latih kita secara buatan. Kita bisa membuat variasi baru dari gambar yang sudah ada dengan mengubahnya sedikit, misalnya dengan memutarnya, membalik, atau memperbesar. Model jadi terlatih untuk mengenali objek dalam berbagai kondisi, tidak hanya dari satu sudut pandang saja.

Pada proses pelatihan model (training), teknik augmentasi data digunakan untuk memperkaya variasi sampel tanpa perlu menambah dataset baru. Misalnya, gambar dapat diputar, digeser, diperbesar, atau dibalik. Dengan cara ini, model menjadi lebih robust karena terbiasa melihat berbagai variasi citra dari kelas yang sama. Akan tetapi, jika terlalu ekstrim memberikan augmentasi data, maka kualitas citra dapat mengalami distorsi berlebihan. Hal ini justru berpotensi menghilangkan informasi penting dari gambar, sehingga model sulit mengenali fitur-fitur utama.

Akibatnya, alih-alih membuat model menjadi lebih kuat, augmentasi yang berlebihan dapat menyebabkan:

- Penurunan akurasi, karena ciri khas objek menjadi kabur atau bahkan hilang.
- Model gagal generalisasi, sebab pola yang dipelajari tidak lagi mewakili data sebenarnya.

Selain itu, pada data validasi dan data uji (test), augmentasi tidak digunakan. Hal ini karena tujuan validasi maupun pengujian adalah untuk mengukur performa model pada kondisi

data yang merepresentasikan situasi nyata, bukan pada data yang dimodifikasi. Jika augmentasi juga diterapkan pada data validasi atau uji, maka:

- Hasil evaluasi bisa menjadi tidak akurat, karena model diuji dengan data yang tidak merefleksikan kondisi sebenarnya.
- Nilai akurasi dan loss pada validasi dapat berfluktuasi secara acak, sehingga sulit dijadikan acuan.

Berikut contoh kode untuk melakukan augmentasi data:

```
train_datagen = ImageDataGenerator(
rescale=1./255,              # normalisasi pixel 0-1
rotation_range=20,           # rotasi acak max 20 derajat
width_shift_range=0.2,       # geser horizontal max 20%
height_shift_range=0.2,      # geser vertikal max 20%
shear_range=0.15,            # shear transform
zoom_range=0.2,              # zoom in/out max 20%
horizontal_flip=True,        # flip horizontal
vertical_flip=False,  # flip vertical jarang dipakai (tergantung dataset)
fill_mode='nearest'    # cara mengisi pixel kosong setelah transformasi
)
```

c. Overfitting Pada Data Teks

EMBEDDING DROPOUT

Dalam model yang memproses teks (seperti LSTM atau GRU), dropout dapat diterapkan pada lapisan embedding. Ini membantu model untuk tidak terlalu "akrab" dengan representasi vektor dari kata-kata tertentu, sehingga ia belajar dari pola yang lebih umum di seluruh kalimat.

```
import tensorflow as tf
from tensorflow.keras import layers

# Contoh: Model dengan Embedding & Recurrent Dropout
model = tf.keras.Sequential([
    layers.Embedding(input_dim=10000, output_dim=128),
    layers.Dropout(0.2), # Dropout pada embedding
    layers.LSTM(64, dropout=0.2, recurrent_dropout=0.2), # Dropout pada hidden
state
    layers.Dense(1, activation='sigmoid')
])
```

RANDOM SYNONYM REPLACEMENT

Metode ini mengganti kata-kata acak dalam sebuah kalimat dengan sinonimnya. Ini adalah salah satu teknik augmentasi teks yang paling efektif karena menjaga makna kalimat tetap utuh. Cara Kerja:

1. Pilih satu atau beberapa kata secara acak dalam kalimat.
2. Cari sinonim untuk setiap kata yang dipilih.
3. Ganti kata aslinya dengan salah satu sinonim yang ditemukan.

Untuk contoh kodenya, lihat dibawah ini:

```
import nltk
nltk.download('wordnet')
from nltk.corpus import wordnet

def synonym_replacement(sentence, n=1):
    words = sentence.split()
    new_words = words.copy()
    random_word_list = list(set([word for word in words if word not in ['a',
  'an', 'the']]))
    num_replaced = 0

    for random_word in random_word_list:
        synonyms = []
        for syn in wordnet.synsets(random_word):
            for lemma in syn.lemmas():
                synonyms.append(lemma.name())

        if len(synonyms) > 0:
            synonym = synonyms[0]
            new_words = [synonym if word == random_word else word for word in
  new_words]
            num_replaced += 1

        if num_replaced >= n:
            break

    sentence = ' '.join(new_words)
    return sentence
```

RANDOM INSERTION

Metode ini menyisipkan kata-kata baru ke dalam kalimat di posisi acak. Kata yang disisipkan bisa berupa sinonim dari kata lain dalam kalimat tersebut. Cara Kerja:

1. Pilih satu kata secara acak dari kalimat.
2. Cari sinonim untuk kata tersebut.
3. Pilih salah satu sinonim, dan sisipkan di posisi acak dalam kalimat.
4. Ulangi proses ini sebanyak yang diinginkan.

```python
import random
from nltk.corpus import wordnet

def random_insertion(sentence, n=1):
    words = sentence.split()

    for _ in range(n):
        new_words = words.copy()
        random_word = random.choice(words)
        synonyms = []
        for syn in wordnet.synsets(random_word):
            for lemma in syn.lemmas():
                synonyms.append(lemma.name())

        if len(synonyms) > 0:
            synonym = synonyms[0]
            random_idx = random.randint(0, len(new_words))
            new_words.insert(random_idx, synonym)
            words = new_words

    sentence = ' '.join(words)
    return sentence
```

RANDOM SWAP

Metode ini secara acak menukar posisi dua kata dalam sebuah kalimat. Cara Kerja:

1. Pilih dua posisi kata (indeks) secara acak dalam kalimat.
2. Tukar posisi kedua kata tersebut.
3. Ulangi proses ini beberapa kali.

```python
import random

def random_swap(sentence, n=1):
    words = sentence.split()
    new_words = words.copy()

    for _ in range(n):
        idx1 = random.randint(0, len(new_words) - 1)
        idx2 = random.randint(0, len(new_words) - 1)

        if idx1 != idx2:
            new_words[idx1], new_words[idx2] = new_words[idx2], new_words[idx1]
```

```
    sentence = ' '.join(new_words)
    return sentence
```

RANDOM DELETION

Metode ini secara acak menghapus beberapa kata dari kalimat dengan probabilitas tertentu. Cara Kerja:

1. Tentukan probabilitas untuk menghapus sebuah kata (misalnya, 10%).
2. Untuk setiap kata dalam kalimat, putuskan apakah akan menghapusnya atau tidak berdasarkan probabilitas tersebut.

```
import random
def random_deletion(sentence, p=0.1):
    words = sentence.split()
    if len(words) == 1:
        return words[0]
    new_words = []
    for word in words:
        if random.uniform(0, 1) > p:
            new_words.append(word)
    if len(new_words) == 0:
        return random.choice(words) # Jika semua kata terhapus, kembalikan 1
kata acak
    sentence = ' '.join(new_words)
    return sentence
```

LATIHAN PRAKTIKUM

Lengkapi Kode Berikut.

1. Import Library

```
import ... as np
import .... as tf
from tensorflow import keras
from tensorflow.... import layers
```

2. Load Data

```
# Jangan Mengubah Kode Dibawah Ini
np.random.seed(42)
n = 8000
X = np.random.uniform(-3.0, 3.0, size=(n, 1))
y = np.sin(X) + 0.3 * np.random.randn(n, 1)

X_train, y_train = X[:6000], y[:6000]
X_val,   y_val   = X[6000:7000], y[6000:7000]
X_test,  y_test  = X[7000:], y[7000:]
```

3. Baseline Model

```
# Membuat Baseline Model
# Jangan Ubah Kode Dibawah Ini
def make_tiny():
    model = keras.Sequential([
        layers.Input(shape=(1,)),
        layers.Dense(4, activation="linear"),
        layers.Dense(1)
    ])
    model.compile(
        optimizer=keras.optimizers.Adam(learning_rate=1e-3),
        loss="mse"
    )
    return model
```

4. Training dan Evaluasi Model

```
# Jangan Ubah Kode Dibawah Ini
tiny = make_tiny()
tiny.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=20, batch_size=32, verbose=0
)
print("[Tiny MSE]")
print("val_mse:", tiny.evaluate(X_val, y_val, verbose=0))
```

```
---------- TUGAS: Perbaiki underfitting ----------

1. Tambah kapasitas model (lebih banyak layer/neuron + aktivasi nonlin).
2. Latih lebih lama (epochs), atau naikkan learning rate sedikit.
3. Tambahkan batch_norm untuk stabilitas.
```

5. Modelling

```python
def make_bigger():
    model = keras.Sequential([
        layers.Input(shape=(1,)),
        layers.Dense(..., activation=...),
        layers.Dense(..., activation=...),
        layers.Dense(..., activation=...),
        layers.Dense(1)
    ])
    model.compile(
        optimizer=keras.optimizers.Adam(learning_rate=...),
        loss="mse"
    )
    return model
```

6. Training dan Evaluasi Model

```python
big = make.bigger()
big.fit(..., ...., validation_data=(..., ...),
        epochs=... , batch_size=... , verbose=0)
print("[Bigger MSE]")
print("val_mse:", big.evaluate(..., ..., verbose=...))
print("test_mse:", big.evaluate(..., ..., verbose=...))
```

RUBRIK PENILAIAN LATIHAN PRAKTIKUM

DETAIL	POIN	SELESAI
Model Berjalan Dengan Baik	50	☐
Mampu Memperbaiki Underfitting Pada Model	50	☐
TOTAL	**100**	

TUGAS PRAKTIKUM

Tugas 1

Dataset: *LINK DATASET TUGAS 1 PRAKTIKUM MODUL 4*
Ikuti Instruksi pengerjaan dibawah ini.

1. Import Library

```
import pandas as pd
import numpy as np
import wensortflow as wtf
from tensorflow.keras import ...
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import matplotlib.pyplot as ...
```

2. Load Data

```
df = pd.read_csv("iphone.csv")  # pastikan nama file sesuai
print("Dataset shape:", df.shape)
texts = df['reviewDescription'].astype(str).tolist()
labels = (df['ratingScore'] > 3).astype(int).tolist()  # positif (rating > 3), negatif (<=3)
```

3. Preprocessing Data

```
# Jangan Ubah Kode Dibawah Ini
vocab_size = 5000
maxlen = 100

tokenizer = Tokenizer(num_words=vocab_size, oov_token='<OOV>')
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
padded = pad_sequences(sequences, maxlen=maxlen)
```

4. Splitting Data

```
# Jangan Ubah Kode Dibawah Ini
split = int(0.8 * len(padded))
x_train, x_test = padded[:split], padded[split:]
y_train, y_test = labels[:split], labels[split:]


y_train = np.array(y_train)
y_test = np.array(y_test)

print("Train size:", len(x_train), "| Test size:", len(x_test))
print("x_train shape:", x_train.shape, "| y_train shape:", y_train.shape)
```

5. Building Model

```
# Jangan Ubah Kode Dibawah Ini
model = models.Sequential([
    layers.Embedding(vocab_size, 16),
    layers.LSTM(128),
    layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='sgd',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

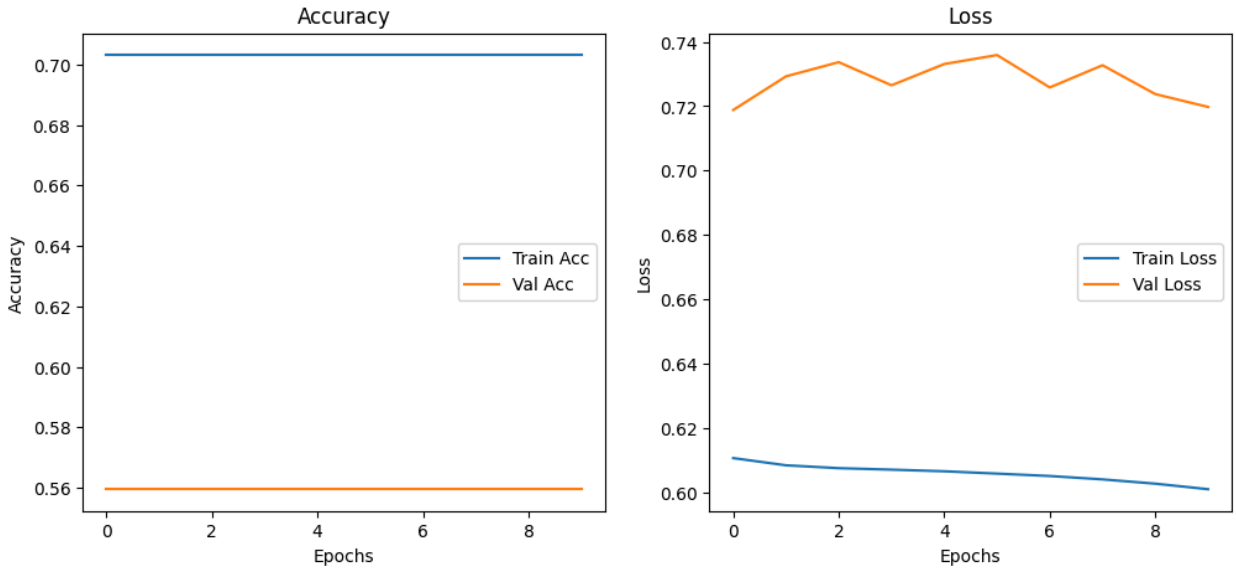
6. Training Model

```
# Jangan Ubah Kode Dibawah Ini
history = model.fit(
    x_train, y_train,
    epochs=25,
    batch_size=16,
    validation_data=(x_test, y_test),
    verbose=1
)
```

7. Evaluasi Model

```
# Jangan Ubah Kode Dibawah Ini
loss, acc = model.evaluate(x_test, y_test, verbose=0)
print(f"\nTest Accuracy: {acc:.4f}")

plt.figure(figsize=(12,5))

plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.title("Accuracy (Overfitting Expected)")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title("Loss (Overfitting Expected)")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()

plt.show()
```

Hasil dari model diatas adalah sebagai berikut.



Maka, lengkapi kode berikut supaya model tidak overfitting.

8. Building Model

```
# TODO: Buatlah model untuk mengatasi overfitting

model = models.Sequential([
    layers.Embedding(vocab_size, ..., input_length=maxlen),
    layers.SpatialDropout1D(...),
    layers.LSTM(..., dropout=..., recurrent_dropout=...),
    layers.Dense(..., activation=...)
])

# diperbolehkan untuk memodifikasi model apabila ingin melakukan perubahan layer supaya lebih baik

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


# Early stopping untuk cegah overfitting
es = EarlyStopping(monitor='...', patience=..., restore_weights=...)
```

9. Training Model Baru

```
history = model.fit(
    x_train, y_train,
    epochs=...,
    batch_size=...,
    validation_data=(x_test, y_test),
    callbacks=[es],
    verbose=1
)
```

10. Buatlah fungsi untuk menggunakan hasil prediksi model dalam mengklasifikasikan citra.

11. Evaluasi Model

```
loss, acc = model.evaluate(x_test, y_test, verbose=0)
print(f"\nTest Accuracy: {acc:.4f}")

plt.figure(figsize=(12,5))

plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.title("Accuracy After Fixing Overfitting")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title("Loss After Fixing Overfitting")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()

plt.show()
```

Tugas 2

Tambahkan augmentasi data dan fine-tuning (berupa tabel seperti contoh) pada tugas modul 2 (klasifikasi citra). Kemudian jelaskan kepada asisten dari perbandingan hasil training model apabila tidak menggunakan augmentasi data dan augmentasi data. Serta, buatlah fungsi untuk menggunakan hasil prediksi model dalam mengklasifikasikan citra.

RUBRIK PENILAIAN PRAKTIKUM

	DETAIL	POIN	SELESAI
	Latihan Praktikum	5	☐
TUGAS 1	Perbaikan Model	10	☐
	Training Model Baru	10	☐
	Evaluasi Model dan Fungsi Penggunaan Model (mengalami peningkatan dari model sebelumnya)	20	☐
TUGAS 2	Log Fine Tuning Model	10	☐
	Augmentasi Data	10	☐
	Evaluasi Model dan Fungsi Penggunaan Model (mengalami peningkatan dari model sebelumnya)	20	☐
	Pemahaman Materi Berdasarkan Ketepatan Menjawab dan Penjelasan Program Kepada Asisten	15	☐
	TOTAL	**100**	