

VERSI 1.2  
AGUSTUS 2025



# [PEMBELAJARAN MESIN]

MODUL 2 - KLASIFIKASI CITRA

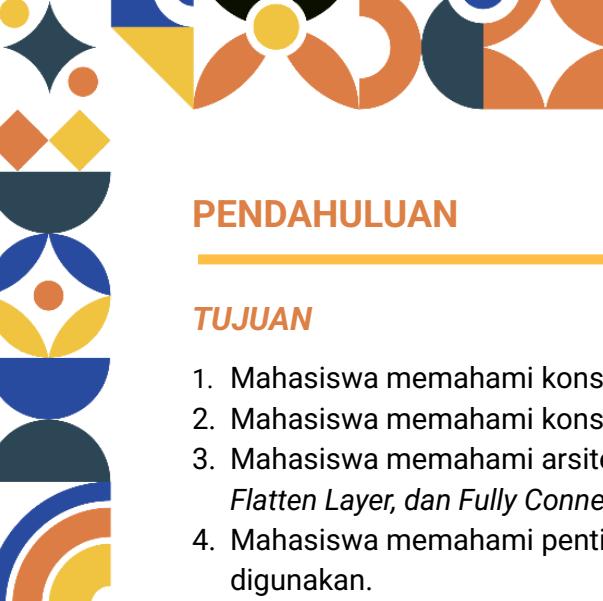
**DISUSUN OLEH:**

YUFIS AZHAR S.KOM, M.KOM

KIARA AZZAHRA

ALVIYA LAELA LESTARI

TIM LABORATORIUM INFORMATIKA  
UNIVERSITAS MUHAMMADIYAH MALANG



## PENDAHULUAN

---

### TUJUAN

1. Mahasiswa memahami konsep dasar citra dan jenis-jenisnya.
2. Mahasiswa memahami konsep klasifikasi citra dan Convolutional Neural Network (CNN)
3. Mahasiswa memahami arsitektur dasar CNN, termasuk *Convolutional Layer, Pooling Layer, Flatten Layer, dan Fully Connected Layer*.
4. Mahasiswa memahami pentingnya preprocessing data citra dan teknik-teknik umum yang digunakan.
5. Mahasiswa mampu membangun dan melatih model CNN menggunakan TensorFlow dan Keras untuk klasifikasi citra.
6. Mahasiswa mampu mengevaluasi model CNN yang telah dilatih.

### TARGET MODUL

1. Mahasiswa mampu meningkatkan kualitas data citra agar lebih bersih dan relevan untuk model.
2. Mahasiswa mampu menyeragamkan data supaya semua citra memiliki format dan ukuran yang konsisten.
3. Mahasiswa mampu membantu proses training model untuk mempelajari pola-pola penting dalam data.
4. Mahasiswa mampu meningkatkan performa model agar lebih akurat dan *robust* (tahan terhadap variasi).

### PERSIAPAN

1. Laptop/PC
2. Google Colaboratory: [COLAB MODUL 2](#)
3. TensorFlow >= 2.10, Keras, matplotlib.pyplot, numpy, os

### DAFTAR ISI

PENDAHULUAN.....	2
TUJUAN.....	2
TARGET MODUL.....	2
PERSIAPAN.....	2
DAFTAR ISI.....	2
<b>MATERI POKOK.....</b>	<b>3</b>
1. CITRA.....	3
2. KLASIFIKASI CITRA.....	4
3. CNN.....	4
1. Convolutional Layer.....	5
2. Pooling Layer.....	6
3. Flatten Layer.....	8





4. Fully Connected Layer.....	9
4. Preprocessing Data.....	11
1. Resizing.....	11
2. Normalisasi.....	12
3. Konversi Tipe Data.....	12
Membuat Model CNN Menggunakan TensorFlow dan Keras.....	12
1. Load Dataset.....	13
2. Importing Library.....	13
3. Preprocessing Data.....	14
4. Build CNN Model.....	15
5. Compile Model.....	18
6. Training Model.....	19
7. Evaluasi Model.....	20
8. Fungsi Klasifikasi Citra.....	21
<b>LATIHAN PRAKTIKUM.....</b>	<b>22</b>
<b>TUGAS PRAKTIKUM.....</b>	<b>26</b>
<b>RUBRIK PENILAIAN.....</b>	<b>26</b>

---

## MATERI POKOK

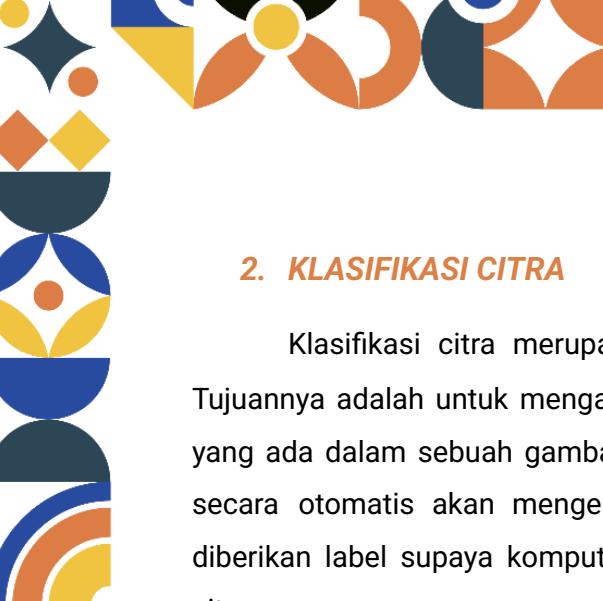
Google Colaboratory: [COLAB MODUL 2](#)

### 1. CITRA

Citra atau yang lebih umumnya disebut gambar adalah representasi visual dari objek atau scene yang dapat berupa gambar, foto atau grafik. Contohnya seperti foto kucing, pemandangan, atau hasil rontgen di rumah sakit. Lalu, bagaimana cara komputer melihat citra? Komputer tidak bisa melihat citra seperti mata manusia. Jadi, komputer menyimpan citra dalam bentuk kumpulan **piksel**. Semakin banyak piksel dalam suatu gambar, semakin jelas dan tajam gambar tersebut.

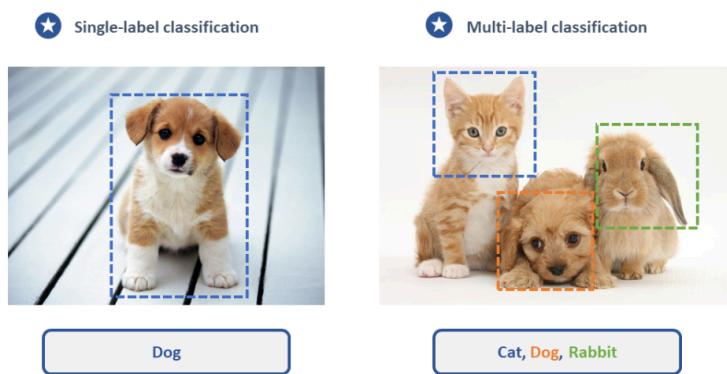
Citra dibagi menjadi dua jenis yaitu, citra berwarna dan citra abu-abu. Citra berwarna memiliki piksel yang menyimpan informasi warna merah, hijau, dan biru (RGB). Gabungan dari ketiga warna ini menghasilkan jutaan warna lain. Sedangkan citra abu-abu atau biasa disebut *grayscale* hanya memiliki warna hitam, putih, dan gradasi abu-abu. Masing-masing pikselnya tidak menyimpan warna, melainkan hanya tingkat kecerahan dari gelap ke terang. Citra ini sering digunakan dalam analisis data karena lebih sederhana untuk proses oleh komputer.





## 2. KLASIFIKASI CITRA

Klasifikasi citra merupakan salah satu tugas utama dalam bidang *computer vision*. Tujuannya adalah untuk mengajarkan komputer untuk mengenali dan mengelompokkan objek yang ada dalam sebuah gambar. Contohnya, ketika kalian melihat foto seekor anjing, otakmu secara otomatis akan mengenali bahwa itu adalah anjing. Jadi, citra akan diproses dan diberikan label supaya komputer dapat mengenali dan mengklasifikasikan objek pada suatu citra.



Klasifikasi *single-label* adalah metode yang mengklasifikasikan suatu objek pada satu kategori saja. Contoh, ketika kalian melakukan klasifikasi pada citra kucing vs anjing. Model harus memilih salah satu dari dua label, yaitu “kucing” atau “anjing”. Tidak mungkin kalau satu foto itu dilabeli “kucing” dan “anjing” secara bersamaan. Intinya, hasil dari klasifikasi *single-label* hanya memiliki satu *output* yang benar. Sedangkan klasifikasi *multi-label* memungkinkan sebuah objek untuk memiliki lebih dari satu kategori atau label secara bersamaan.

## 3. CNN

*Convolutional Neural Network* (CNN) adalah salah satu jenis *neural network* (jaringan saraf tiruan) yang dirancang khusus untuk memproses data visual, seperti citra. CNN menjadi salah satu metode yang paling berhasil dalam bidang *computer vision* karena kemampuannya dalam mengenali pola-pola visual secara otomatis, tanpa perlu kita beritahu secara spesifik apa yang harus dicari. Sederhananya, CNN terinspirasi dari cara kerja otak manusia sehingga arsitektur ini bekerja seperti mata dan otak kita saat melihat suatu objek. Mata manusia tidak





memproses seluruh gambar sekaligus, melainkan mengenali sebuah objek dengan mengidentifikasi fitur-fitur kecilnya terlebih dahulu—seperti garis, sudut, atau warna, lalu menggabungkan fitur-fitur tersebut menjadi bentuk yang lebih kompleks.

CNN memiliki beberapa lapisan-lapisan utama dengan fungsi spesifik yang memproses data dari lapisan sebelumnya.

### 1. Convolutional Layer

Lapisan konvolusi atau disebut *convolutional layer* adalah lapisan inti dari CNN yang berguna untuk **mengekstrak fitur** pada gambar. Cara kerjanya adalah menggunakan **filter/kernel**, yaitu sebuah matriks kecil berisi angka-angka. Filter/kernel ini bertugas untuk mencari pola-pola spesifik dalam gambar, seperti garis tepi, sudut, atau tekstur. Hasil dari lapisan ini disebut dengan peta fitur (*feature map*).

Dalam CNN, data citra pada umumnya direpresentasikan dalam tiga dimensi: **Tinggi (Height), Lebar (W), dan Jumlah Kanal (C)**. Sebagai contoh, sebuah citra RGB (3 kanal warna) berukuran 5x5 piksel akan memiliki dimensi input (5x5x3), dimana 3 adalah jumlah kanal warnanya. Untuk memproses citra ini, CNN menggunakan sebuah filter/kernel yang juga memiliki dimensi tiga dimensi, yaitu tinggi (K), lebar (K), dan jumlah kanal (C). Penting untuk dicatat juga bahwa jumlah kanal harus sama dengan jumlah kanal pada citra input. Sebagai contoh, untuk citra RGB (5 x 5 x 3), ukuran kernel yang umum digunakan adalah (3 x 3 x 3).

Secara umum, proses lapisan *convolution* dapat dirumuskan sebagai berikut:

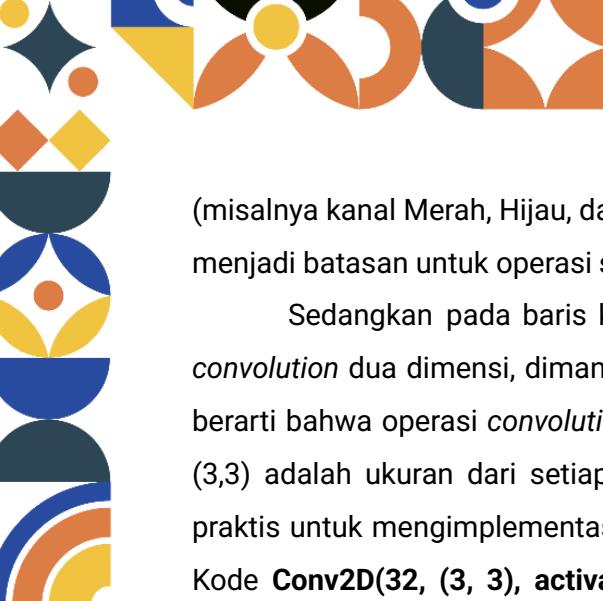
$$feature\_surface_{out} = f \left( \sum_{i=1}^3 M_i * W_i + B \right)$$

Pada rumus tersebut,  $M_i$  merepresentasikan ukuran fitur pada input,  $W_i$  adalah *weight* matriks dari kernel konvolusi,  $f(\cdot)$  adalah fungsi aktivasi non linier (misalnya ReLU) dan  $feature\_surface_{out}$  adalah output dari fitur. Jika kita menggunakan pustaka Keras untuk membuat layer konvolusi, kodennya adalah sebagai berikut :

```
Input(shape=(H, W, 3)),  
Conv2D(32, (3, 3), activation='relu'),
```

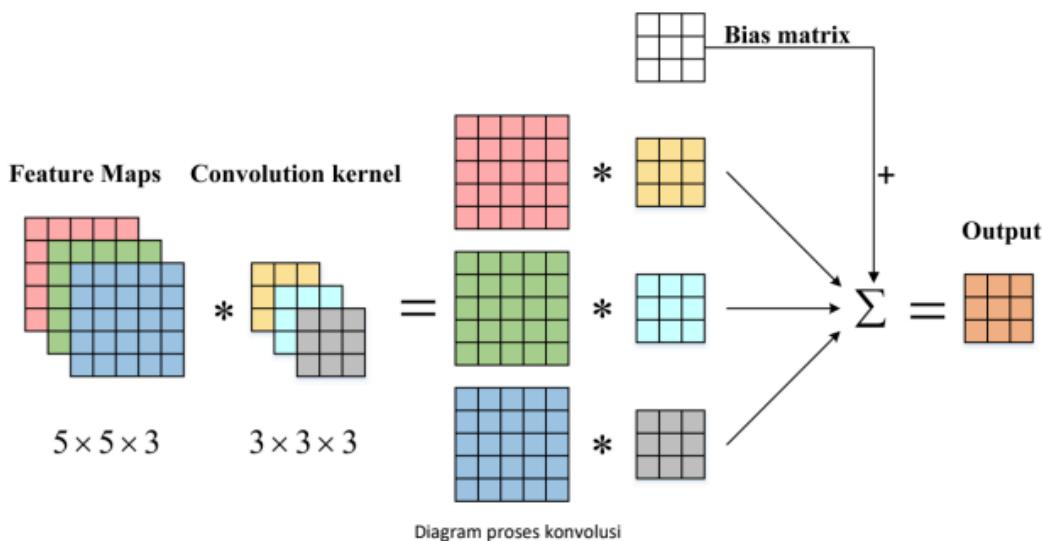
`Input(shape=H,W,3)` adalah definisi dari citra input kita. Angka 3 di sini merujuk pada jumlah kanal warna (C) pada citra, yang biasanya untuk citra RGB (Red, Green, Blue). Ini sesuai dengan komponen  $M_i$  dan **C** dalam rumus. Di mana  $M_i$  adalah data fitur pada setiap kanal input





(misalnya kanal Merah, Hijau, dan Biru), dan C adalah jumlah total kanal (dalam hal ini, 3), yang menjadi batasan untuk operasi **sum** ( $\Sigma$ ) pada rumus.

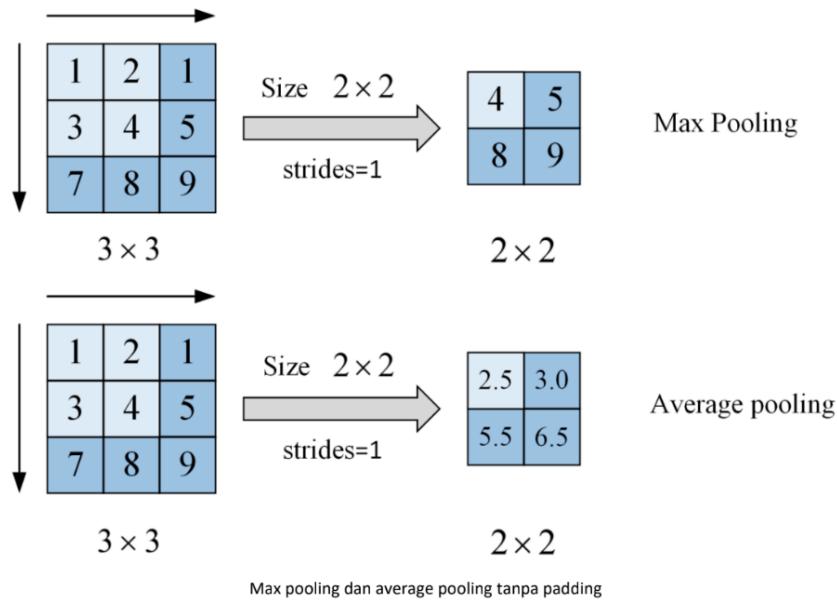
Sedangkan pada baris kedua, fungsi Conv2D adalah fungsi yang menjalankan operasi *convolution* dua dimensi, dimana 32 adalah jumlah filter atau kernel yang akan digunakan yang berarti bahwa operasi *convolution* akan menghasilkan 32 *featured maps*. Selanjutnya parameter (3,3) adalah ukuran dari setiap kernel. Jadi, secara singkat, baris kode tersebut adalah cara praktis untuk mengimplementasikan seluruh proses matematika yang dijelaskan dalam rumus. Kode **Conv2D(32, (3, 3), activation='relu')** secara otomatis akan melakukan perkalian antara input ( $M_i$ ) dengan *kernel* ( $W_i$ ), menjumlahkan hasilnya ( $B$ ), dan mengaktifkannya dengan fungsi ( $f$ ), lalu mengulanginya sebanyak 32 kali untuk menghasilkan 32 Peta Fitur. Untuk gambarannya, kalian coba amati diagram berikut:



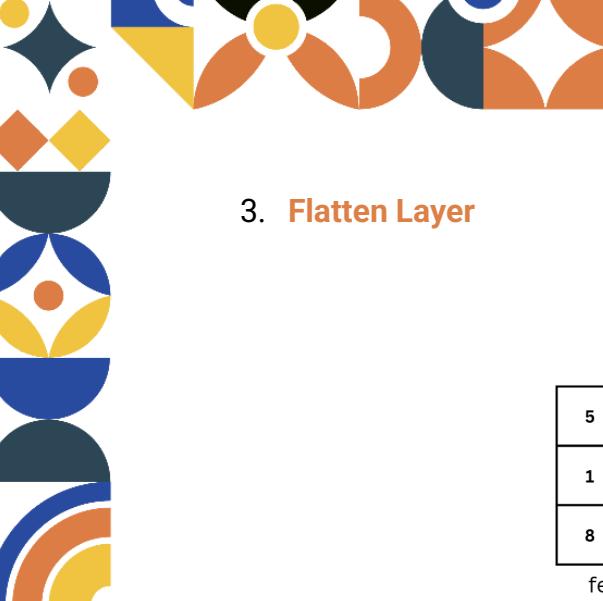
## 2. Pooling Layer

Lapisan *pooling* akan menerima *feature map* dari lapisan sebelumnya. Tugas dari pooling layer adalah untuk menyederhanakan informasi yang ada (*down sampling*). Lapisan ini mengurangi ukuran *feature map* dengan mengambil nilai-nilai terpenting dari *feature map* supaya hanya esensi dari fitur yang dipertahankan, sehingga lapisan *pooling* akan membuat model menjadi lebih efisien dalam hal komputasi dan membantunya menjadi tidak terlalu sensitif terhadap pergeseran kecil pada posisi objek dalam gambar.

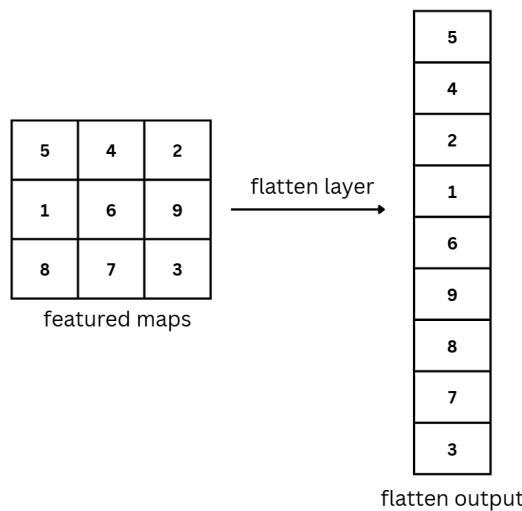




Pada umumnya, jenis *pooling layer* yang sering digunakan adalah **max pooling** dan **average pooling**. Singkatnya, **max pooling** adalah jenis *pooling* yang mengambil nilai maksimum dari setiap jendela (window) kecil yang bergeser di seluruh *featured maps*, hal ini akan membuat model menjadi lebih tidak sensitif terhadap pergeseran kecil (translasi) pada posisi objek. Sedangkan **average pooling** adalah jenis *pooling layer* yang menghitung nilai rata-rata dari setiap jendela kecil yang bergeser di seluruh *featured maps*. Tidak seperti *max pooling* yang berfokus pada fitur paling dominan, *average pooling* memberikan gambaran umum atau rata-rata dari fitur di setiap area. Metode ini cocok digunakan ketika detail yang terdistribusi secara merata lebih penting daripada fitur tunggal yang menonjol. Akan tetapi, kedua jenis *pooling* ini bukan berarti yang terbaik, ada jenis *pooling* lain yang lebih baik, namun jenis tersebut adalah jenis yang paling umum digunakan dalam CNN.



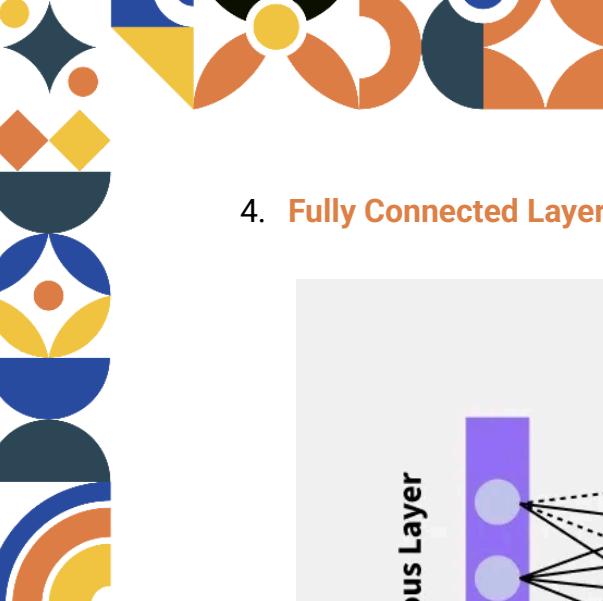
### 3. Flatten Layer



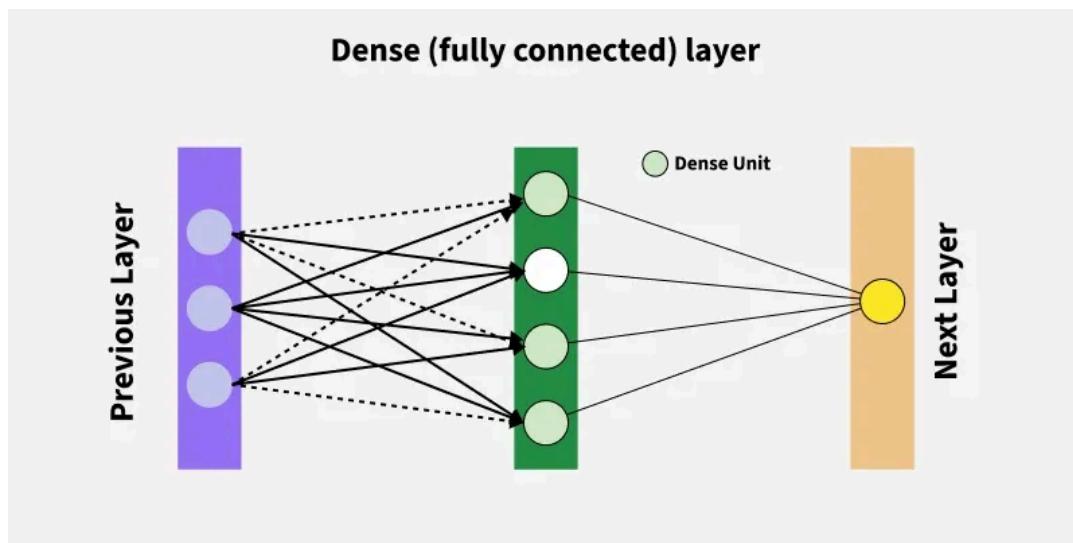
Setelah melewati *convolutional layer* dan *pooling layer*, dimana fitur-fitur yang lebih kompleks terus-menerus dibangun, informasi yang sudah diringkas kemudian diubah menjadi satu vektor data panjang melalui *flatten layer*. Jadi, lapisan *flatten* akan menyiapkan data untuk tahap akhir berupa vektor. Vektor data ini, yang sekarang berisi representasi numerik dari semua fitur penting yang telah diekstrak, kemudian dimasukkan ke lapisan selanjutnya.

Contohnya, ketika kita memiliki sebuah *featured maps* dari *pooling layer* dengan ukuran  $3 \times 3 \times 5$ , total elemen atau numerik didalamnya adalah  $3 \times 3 \times 5 = 45$ . Ketika data ini melewati lapisan *flatten*, data tersebut akan diubah menjadi sebuah vektor satu dimensi dengan panjang 45. Jadi, bisa dibilang Lapisan *Flatten* adalah tahap "persiapan" data. Ia mengambil semua informasi fitur yang telah ditemukan oleh CNN dan menyajikannya dalam format yang bisa dipahami oleh lapisan klasifikasi, memungkinkan model untuk membuat prediksi akhir yang akurat.





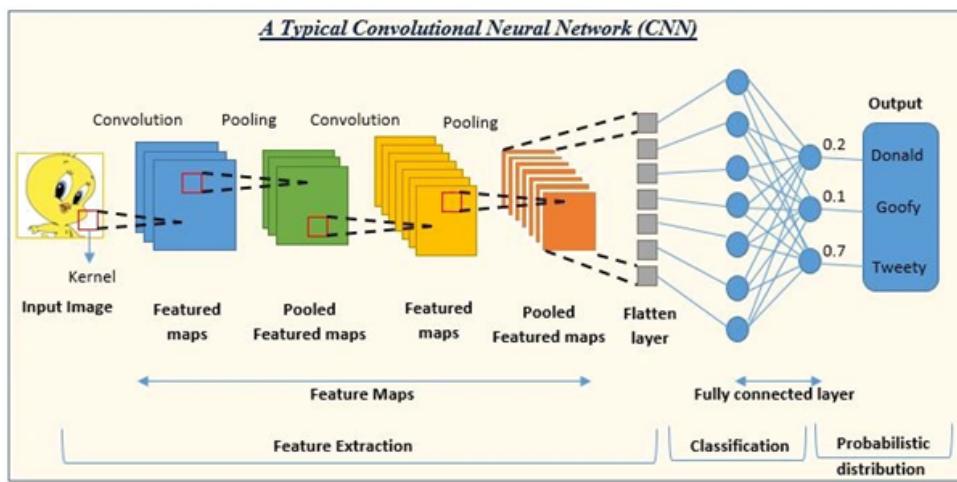
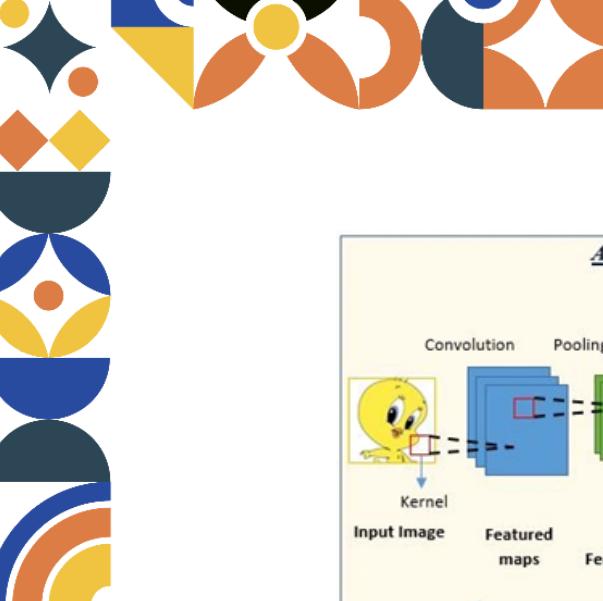
#### 4. Fully Connected Layer



Lapisan *fully connected* atau biasa disebut ***dense layer*** adalah lapisan yang akan menganalisis hubungan di antara semua fitur yang ada untuk membuat keputusan akhir. Lapisan ini hanya menerima input dalam bentuk satu dimensi, sehingga *flatten layer* sangat diperlukan dalam model CNN. Analoginya, jika *flatten layer* adalah jembatan, maka *fully connected layer* adalah otak yang membuat keputusan klasifikasi. Dengan mengkombinasikan semua fitur yang telah diekstrak, Lapisan *fully connected* menggunakan bobot dan bias yang telah dilatih untuk mengidentifikasi pola-pola kompleks dan membuat keputusan klasifikasi. Output dari lapisan ini, terutama di bagian akhir arsitektur CNN, sering kali berupa probabilitas untuk setiap kelas, yang menunjukkan seberapa besar keyakinan model terhadap setiap kategori yang mungkin, seperti "kucing" atau "anjing".

Untuk lebih jelasnya, coba amati diagram CNN dibawah ini.





Pada diagram di atas, *input image* yang diberikan adalah citra "tweety". Citra kemudian akan diproses melalui lapisan *convolutional* untuk diekstrak fitur-fitur yang ada pada citra. Di lapisan ini, sebuah matriks kecil yang disebut **kernel/filter** akan menggeser di seluruh gambar. Setiap kali kernel/filter bergeser, ia akan melakukan perhitungan yang menghasilkan satu nilai. Nilai-nilai ini dikumpulkan menjadi sebuah *feature map*.

Selanjutnya *feature map* akan dibawa ke *pooling layer* untuk dirangkum isi dari data-data yang ada pada *feature map*. Hasilnya biasa disebut dengan *pooled featured maps*. Ukurannya lebih kecil tapi esensi dari fitur-fitur tetap terjaga.

Pada diagram diatas, setelah diberikan *pooling layer*, diberikan kembali *convolutional* dan *pooling layer* lagi. Dua lapisan ini biasanya memang diulang beberapa kali, tujuannya adalah untuk memproses ulang hasil dari lapisan sebelumnya, dimana fitur-fitur sederhana digabungkan menjadi fitur yang lebih kompleks di setiap tahapan.

Setelah proses ekstraksi fitur selesai, hasilnya masuk ke tahap klasifikasi. Di sini, *feature maps* yang terakhir akan diubah menjadi satu baris data panjang melalui Lapisan *Flatten*. Data ini kemudian dimasukkan ke Lapisan *Fully Connected*, yang bertugas menggabungkan semua informasi fitur yang telah diekstrak. Lapisan ini akan mempelajari pola dari fitur-fitur tersebut untuk membuat keputusan akhir. Keputusan ini direpresentasikan di Lapisan *Output* dalam bentuk probabilitas. Setiap *neuron* di lapisan ini mewakili sebuah kategori, dan nilai probabilitasnya menunjukkan seberapa besar keyakinan model bahwa gambar tersebut termasuk dalam kategori itu.

Sebagai contoh dari diagram, setelah melewati seluruh proses, model memprediksi bahwa gambar tersebut adalah "Tweety" dengan probabilitas 70%, yang merupakan keyakinan



tertinggi. Alur bertahap ini memungkinkan CNN untuk "belajar" dan "melihat" gambar dengan cara yang mirip dengan manusia, dari fitur-fitur kecil hingga pengenalan objek secara keseluruhan.

Untuk mengetahui lebih jelasnya lagi, tonton video berikut: [LINK VIDEO CNN](#)

#### 4. Preprocessing Data

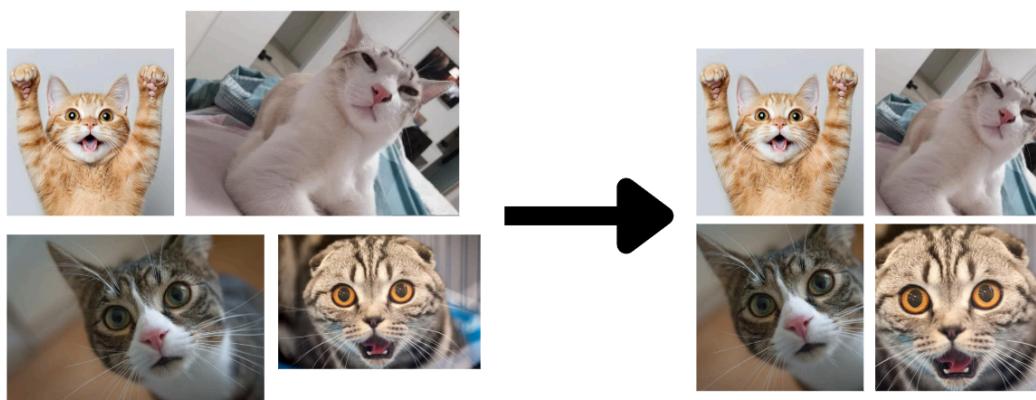
Di mata kuliah penggalian data, pasti kalian sudah tidak asing dengan istilah preprocessing (pra-pemrosesan) data. Tidak hanya data teks, data citra juga dapat dilakukan pre-processing. Tujuannya adalah supaya:

- Meningkatkan kualitas data citra agar lebih bersih dan relevan untuk model
- Menyeragamkan data supaya semua citra memiliki format dan ukuran yang konsisten
- Membantu proses training model untuk mempelajari pola-pola penting dalam data
- Meningkatkan performa model agar lebih akurat dan robust (tahan terhadap variasi)

Berikut adalah beberapa teknik preprocessing yang paling umum dan sering digunakan.

##### 1. Resizing

Model CNN biasanya memiliki ekspektasi ukuran input yang tetap. Jika dataset kita memiliki citra dengan berbagai ukuran, kita perlu mengubah ukurannya menjadi dimensi yang sama. Misalnya, kita mungkin mengubah semua gambar menjadi 224x224 piksel atau 128x128 piksel, tergantung kebutuhan kalian.



Cara implementasi ke kodingannya seperti kode dibawah ini.



```
resized_image = tf.image.resize(image, (128, 128))
```

## 2. Normalisasi

Normalisasi adalah proses mengubah rentang nilai piksel dalam citra menjadi rentang yang lebih standar, biasanya antara 0 dan 1, atau -1 dan 1. Ini akan membantu mempercepat proses *training* model dan mencegah satu piksel dengan nilai yang sangat tinggi mendominasi proses pembelajaran.

```
normalized_image = resized_image / 255.0
```

## 3. Konversi Tipe Data

Sebelum masuk ke model, data citra perlu memiliki tipe data numerik yang tepat. Secara umum, data gambar yang dinormalisasi (rentang 0-1) sebaiknya menggunakan *floating-point* untuk memastikan akurasi perhitungan matematis.

```
final_image = tf.cast(normalized_image, dtype=tf.float32)
```

# Membuat Model CNN Menggunakan TensorFlow dan Keras

Agar lebih interaktif, klik link colab berikut: [COLAB MODUL 2](#)

Jika data yang kita miliki adalah data citra, framework yang paling sering digunakan dan direkomendasikan adalah TensorFlow dengan library Keras. Karena, tensorflow berguna sebagai "mesin" yang akan menjalankan semua perintah yang kita berikan melalui Keras, seperti perkalian matriks, operasi konvolusi, dan optimasi weight pada model. Sedangkan Keras akan berperan untuk menyembunyikan kompleksitas matematika dan komputasi di balik layar, sehingga kita bisa fokus pada arsitektur model, seperti mendefinisikan lapisan-lapisan (misalnya, Conv2D, MaxPooling2D, Flatten, dan Dense). Jika kalian menggunakan diluar google colab, pastikan tensorflow sudah terinstal, jika belum gunakan command dibawah:

```
! pip install tensorflow
```



## 1. Load Dataset

Langkah selanjutnya, pastikan dataset sudah kalian install. Kalian bisa menginstall secara langsung dari *google colab* atau secara manual. Akses [LINK DATASET MATERI MODUL 1](#) agar kalian bisa mengetahui secara langsung materi dibawah. (Jika tidak, maka jangan run kode yang ada di bawah ini supaya outputnya tidak hilang).

## 2. Importing Library

Jika data yang kita miliki adalah data citra, framework yang paling sering digunakan dan direkomendasikan adalah TensorFlow dengan library Keras. Karena, tensorflow berguna sebagai "mesin" yang akan menjalankan semua perintah yang kita berikan melalui Keras, seperti perkalian matriks, operasi konvolusi, dan optimasi weight pada model. Sedangkan Keras akan berperan untuk menyembunyikan kompleksitas matematika dan komputasi di balik layar, sehingga kita bisa fokus pada arsitektur model, seperti mendefinisikan lapisan-lapisan (misalnya, Conv2D, MaxPooling2D, Flatten, dan Dense).

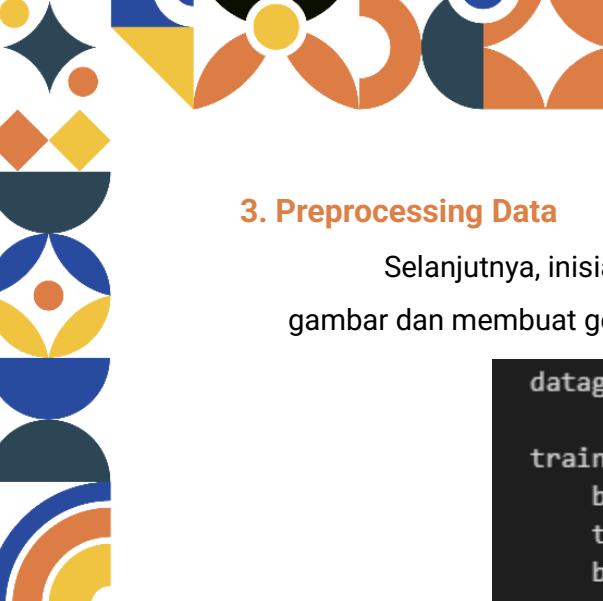
```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
```

- `ImageDataGenerator` akan berfungsi untuk melakukan augmentasi data citra dan untuk memuat data citra dari direktori ke dalam model dengan format yang benar.
- `Sequential` adalah model API yang paling sederhana dan paling umum digunakan. Jadi, `sequential` akan berguna untuk membuat model jaringan saraf (*neural network*) dengan menumpuk beberapa *layer* satu per satu secara linear. Ibaratnya `sequential` akan bertindak sebagai wadah untuk membangun modelnya.

Jika sudah, buat variabel untuk menyimpan path dataset kalian.

```
base_dir = 'animals'
```





### 3. Preprocessing Data

Selanjutnya, inisialisasikan **ImageDataGenerator** untuk melakukan *preprocessing* gambar dan membuat generator data *train* dan *valid*.

```
datagen = ImageDataGenerator(rescale=1./255)

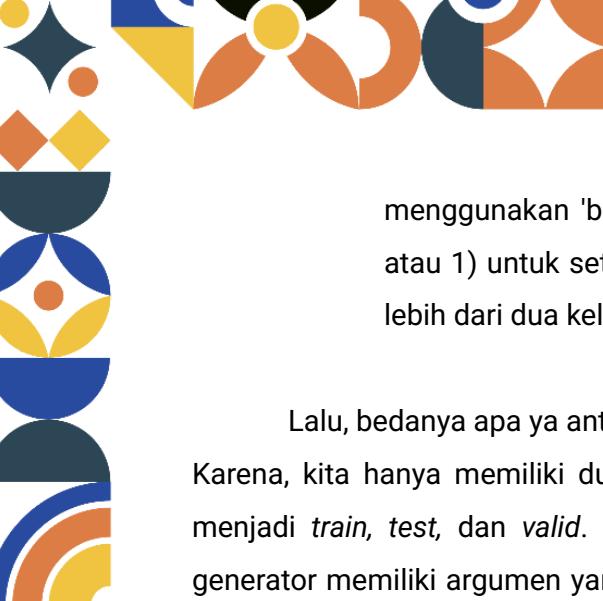
train_generator = datagen.flow_from_directory(
    base_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary')

validation_generator = datagen.flow_from_directory(
    base_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary')
```

Penjelasan argumen pada *train\_generator* dan *validation\_generator*:

- **base\_dir**: Ini adalah argumen pertama yang diberikan, yaitu variabel yang menunjuk ke direktori utama tempat dataset gambar berada ('animals'). *flow\_from\_directory* akan mencari subdirektori di dalam *base\_dir*. Dalam kasus dataset ini, ada subdirektori 'cat' dan 'dog'. Metode ini secara otomatis akan menganggap nama subdirektori sebagai nama kelas.
- **target\_size=(150, 150)**: Argumen ini menentukan ukuran (tinggi, lebar) di mana semua gambar akan diubah ukurannya. Meskipun gambar asli mungkin memiliki resolusi yang berbeda-beda, generator akan mengubah ukurannya menjadi 150 x 150 piksel sebelum memasukkannya ke dalam model. Ini penting karena model CNN memerlukan input gambar dengan ukuran yang konsisten.
- **batch\_size=32**: Ini mengatur ukuran batch data yang akan dihasilkan oleh generator. Selama pelatihan, model tidak memproses semua gambar sekaligus, melainkan memprosesnya dalam kelompok-kelompok kecil yang disebut batch. Ukuran batch 32 berarti setiap kali model meminta data dari generator, generator akan memberikan 32 gambar dan label yang sesuai.
- **class\_mode='binary'**: Argumen ini menentukan jenis label yang akan dihasilkan. Karena kita melakukan klasifikasi biner (pizza vs not pizza, yaitu 2 kelas), kita





menggunakan 'binary'. Ini akan membuat generator menghasilkan label biner (0 atau 1) untuk setiap gambar, yang sesuai dengan kelasnya. Kalau kalian memiliki lebih dari dua kelas, maka akan menggunakan 'categorical'.

Lalu, bedanya apa ya antara kedua generator tersebut jika memiliki argumen yang sama? Karena, kita hanya memiliki dua subfolder dataset yaitu '*pizza*' dan '*not pizza*' tanpa dipisah menjadi *train*, *test*, dan *valid*. Sehingga, argumen yang digunakan jadi sama. Nah, jika dua generator memiliki argumen yang sama secara fungsional tidak ada perbedaan dalam sumber data yang mereka akses. Keduanya akan membaca gambar dari direktori yang sama (*base\_dir*), dengan ukuran target, ukuran *batch*, dan mode kelas yang sama. Perbedaan utama terletak pada tujuan penggunaannya dalam proses pelatihan model.

- **train\_generator** digunakan dalam metode `model.fit()` untuk menyediakan batch data yang akan digunakan oleh model untuk belajar dan memperbarui bobotnya.
- **validation\_generator** juga digunakan dalam metode `model.fit()`, tetapi untuk menyediakan batch data yang digunakan untuk memantau kinerja model selama pelatihan tanpa memperbarui bobot model.

Hasil validasi ini membantu mengidentifikasi apakah model mulai mengalami **overfitting**—misalnya, kinerja baik di data *train* tetapi menurun di data *valid*. Namun, jika kedua generator membaca dari direktori yang sama, berarti proses validasi dilakukan pada data yang sama persis dengan data pelatihan. Hal ini dapat memberikan gambaran kinerja validasi yang keliru dan menutupi potensi masalah *overfitting*. Dalam praktik yang tepat, `validation_generator` seharusnya membaca dari set data validasi terpisah yang belum pernah digunakan model saat pelatihan.

#### 4. Build CNN Model

Untuk membuat model CNN, kita perlu memiliki pemahaman konseptual tentang bagaimana model akan memproses gambar seperti yang sudah dijelaskan pada point 3: CNN, tentang lapisan-lapisan yang dibutuhkan dalam membuat model yang akan mengidentifikasi pola-pola visual untuk membedakan kedua citra tersebut. Proses ini sebagian besar dilakukan melalui tahapan *feature extraction* yang terdiri dari kombinasi lapisan *convolution* dan *pooling*. Sebelum membuat modelnya, mari kita bayangkan model kita butuh lapisan apa saja supaya dapat mengklasifikasikan gambar *pizza* dan *not pizza*. Pertama, kita memerlukan *convolution layer* untuk mengekstraksi fitur dasar seperti garis, tepi, atau tekstur dari gambar. Setelah itu



kita butuh *pooling layer* untuk mengecilkan ukuran representasi sambil tetap menjaga informasi penting. Kombinasi *convolution* dan *pooling* ini biasanya diulang beberapa kali agar model bisa mengenali pola yang semakin kompleks, misalnya bentuk khas pizza atau ciri gambar yang bukan pizza.

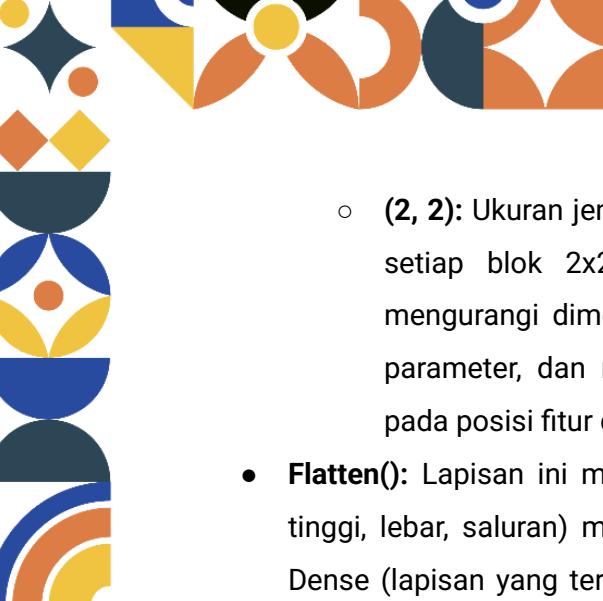
Selanjutnya, hasil ekstraksi fitur tersebut diproses melalui lapisan flatten untuk diubah menjadi vektor satu dimensi, kemudian diteruskan ke dense layer agar model bisa melakukan proses pengambilan keputusan. Terakhir, kita membutuhkan output layer dengan aktivasi sigmoid untuk menghasilkan probabilitas apakah gambar yang diberikan termasuk pizza atau bukan. Jika ditulis dalam bentuk kode, maka akan terlihat seperti kode berikut:

```
model = Sequential([
    Conv2D(16, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.summary()
```

Lapisan-lapisan CNN bisa kalian modifikasi sesuai kebutuhan dataset kalian.

- **Conv2D(32, (3, 3), activation='relu', input\_shape=(150, 150, 3)):** Ini adalah lapisan pertama dalam model, yaitu lapisan konvolusi 2D (Conv2D).
  - **32:** Jumlah filter (kernel) konvolusi dalam lapisan ini. Setiap filter akan belajar untuk mendeteksi fitur yang berbeda dalam gambar.
  - **(3, 3):** Ukuran setiap filter konvolusi (tinggi x lebar). Di sini, filter berukuran 3x3 piksel.
  - **activation='relu':** Fungsi aktivasi yang digunakan setelah operasi konvolusi. ReLU (*Rectified Linear Unit*) adalah fungsi aktivasi yang umum digunakan dan efektif.
  - **input\_shape=(150, 150, 3):** Ini **hanya perlu ditentukan pada lapisan pertama** model Sequential. Ini mendefinisikan bentuk input yang diharapkan oleh model: tinggi 150 piksel, lebar 150 piksel (sesuaikan dengan ukuran data citra kalian), dan 3 saluran warna (untuk gambar RGB).
- **MaxPooling2D((2, 2)):** Ini adalah lapisan max pooling 2D.



- **(2, 2):** Ukuran jendela pooling. Lapisan ini mengambil nilai piksel maksimum dari setiap blok 2x2 piksel dalam output lapisan sebelumnya. Ini membantu mengurangi dimensi spasial (tinggi dan lebar) dari output, mengurangi jumlah parameter, dan membantu model menjadi lebih robust terhadap variasi kecil pada posisi fitur dalam gambar.
- **Flatten():** Lapisan ini mengubah output dari lapisan sebelumnya (yang berbentuk 3D: tinggi, lebar, saluran) menjadi vektor 1D (satu dimensi). Ini diperlukan karena lapisan Dense (lapisan yang terhubung penuh) yang mengikutinya mengharapkan input dalam bentuk 1D.
- **Dense(64, activation='relu'):** Ini adalah lapisan Dense (*fully connected*) pertama.
  - **64:** Jumlah unit (neuron) di lapisan ini. Setiap neuron terhubung ke semua neuron di lapisan sebelumnya.
- **Dense(1, activation='sigmoid'):** Ini adalah lapisan Dense terakhir (output layer).
  - **1:** Jumlah unit di lapisan output. Karena ini adalah masalah klasifikasi biner (pizza vs not pizza), kita hanya membutuhkan satu unit output.
  - **activation='sigmoid':** Fungsi sigmoid menghasilkan output antara 0 dan 1, yang dapat diinterpretasikan sebagai probabilitas kelas positif (misalnya, probabilitas bahwa gambar adalah anjing).
- **model.summary():** Metode ini mencetak ringkasan model, termasuk nama setiap lapisan, bentuk outputnya, dan jumlah parameter yang dapat dilatih di setiap lapisan. Ini sangat berguna untuk memahami struktur model dan jumlah parameter yang perlu dipelajari. Sehingga, dari kode diatas, kita dapat melihat gambaran model seperti berikut:

Model: "sequential"		
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 16)	448
max_pooling2d (MaxPooling2D)	(None, 74, 74, 16)	0
conv2d_1 (Conv2D)	(None, 72, 72, 32)	4,640
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 32)	0
flatten (Flatten)	(None, 41472)	0
dense (Dense)	(None, 32)	1,327,136
dense_1 (Dense)	(None, 1)	33

Total params: 1,332,257 (5.08 MB)  
Trainable params: 1,332,257 (5.08 MB)  
Non-trainable params: 0 (0.00 B)





Jadi, sebenarnya tidak ada aturan baku yang menentukan jumlah pasti lapisan yang harus digunakan. Melainkan, jumlah lapisan dalam model CNN sangat bergantung pada beberapa faktor:

- **Kompleksitas masalah:** Klasifikasi pizza dan not pizza adalah contoh kasus yang relatif sederhana dibandingkan dengan klasifikasi ribuan objek berbeda. Untuk masalah yang lebih kompleks, biasanya dibutuhkan banyak lapisan untuk mengekstrak fitur yang lebih halus dan membedakan antar kelas.
- **Ukuran dan kualitas data citra:** Misal data citra yang digunakan berukuran kecil atau memiliki resolusi yang rendah dan banyak noise, model yang terlalu banyak lapisan akan cenderung mengalami *overfitting* (model terlalu menghafal data train dan performanya akan buruk pada data baru). Sebaliknya, jika data yang kita miliki adalah data besar dan berkualitas tinggi, lebih banyak lapisan bisa membantu model belajar representasi yang lebih kaya.
- **Ketersediaan Sumber Daya Komputasi:** Model dengan lebih banyak lapisan membutuhkan lebih banyak daya komputasi dan waktu pelatihan. Jika sumber daya terbatas, arsitektur yang lebih sederhana dengan lebih sedikit lapisan mungkin lebih praktis.

## 5. Compile Model

Supaya model bisa di-*training*, kita harus meng-*compile* (menyiapkan/mengatur model sebelum dilatih):

```
from tensorflow.keras.optimizers import Adam

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

Seperti yang sudah dijelaskan di modul sebelumnya, kita akan menggunakan optimizer Adam. *Loss* adalah fungsi yang akan digunakan oleh model selama *training* untuk mengukur seberapa baik atau seberapa buruk kinerja model pada data *train*. Tujuannya adalah untuk meminimalkan nilai fungsi *loss*. Sedangkan '*binary\_crossentropy*' adalah nama fungsi *loss* spesifik yang digunakan di sini. ***Binary crossentropy*** adalah fungsi kerugian yang sangat umum dan direkomendasikan untuk masalah klasifikasi biner (ketika kalian memiliki dua kelas, seperti pizza vs not pizza). Fungsi ini mengukur perbedaan antara distribusi probabilitas yang diprediksi oleh model (output dari lapisan sigmoid, yaitu probabilitas antara 0 dan 1) dan distribusi





probabilitas yang sebenarnya (label biner 0 atau 1). Semakin kecil nilai `binary_crossentropy`, semakin dekat prediksi model dengan label sebenarnya. Jadi, dengan mengatur `loss='binary_crossentropy'`, kita memberi tahu model untuk menggunakan fungsi ini sebagai ukuran "kesalahan" selama pelatihan, dan proses optimasi (yang ditentukan oleh `optimizer=Adam()`) akan berusaha mengurangi kesalahan ini.

## 6. Training Model

Jika model sudah siap, selanjutnya kita tinggal *training* model dengan dataset yang sudah kita persiapkan sebelumnya. .

```
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

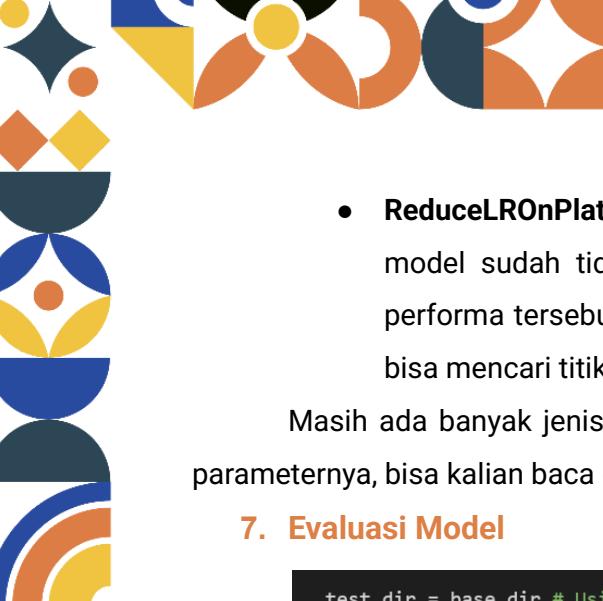
epochs = 7
history = model.fit(
    train_generator,
    epochs=epochs,
    validation_data=validation_generator,
    callbacks=[EarlyStopping(monitor='val_loss', patience=5), ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3,
    verbose=1, min_lr=1e-7)]
)
```

`model.fit()` akan menjalankan *training* menggunakan data yang sudah kita berikan pada parameter yang tertera seperti data *train* (`train_generator`), berapa *epoch*-nya, data *valid* (`validation_generator`), dan `callbacks`. Apa itu `callbacks`? **Callback** adalah proses pelatihan model yang memungkinkan fungsi atau tindakan tertentu pada titik-titik spesifik selama proses pelatihan berlangsung. Atau lebih jelasnya, *callback* adalah fungsi atau objek yang kita kasih ke `model.fit()` supaya si *callback* ini jalan secara otomatis di momen-momen tertentu selama *training* model.

Misalnya, kita memiliki 7 `epochs` untuk melatih model mencapai akurasi tertingginya, tetapi ternyata setelah model di-*training* hingga `epochs` terakhir, model malah mengalami penurunan akurasi. Setelah dilakukan visualisasi analisis, ternyata model mengalami peningkatan akurasi tertinggi pada `epochs` ke-7. Tapi, belum tentu kan kalau kita kurangi `epochs`-nya sesuai pada puncak akurasi, model akan memiliki akurasi yang sama, bahkan akan membuang-buang waktu jika kita *training* ulang model tersebut. Disinilah, *callback* berguna banget. Berikut beberapa contoh *callback* yang umum digunakan:

- **EarlyStopping:** Otomatis memberhentikan *training* jika performa model tidak mengalami peningkatan performa, jadi tidak akan terjadi *overfitting*.
- **ModelCheckpoint:** Menyimpan bobot terbaik ketika performa model berada di puncak maksimal, meskipun *training*-nya tetap berjalan sampai akhir.





- **ReduceLROnPlateau:** Berguna untuk menurunkan *learning rate* jika performa model sudah tidak bisa membaik. Ibaratnya, ketika model sudah stuck pada performa tersebut, fungsi ini akan memberi *learning rate* yang lebih kecil supaya bisa mencari titik optimalnya lagi.

Masih ada banyak jenis *callback* yang lainnya. Untuk lebih lengkap beserta penjelasan parameternya, bisa kalian baca dokumentasi berikut: [API CALLBACK KERAS](#)

## 7. Evaluasi Model

```
test_dir = base_dir # Using the same directory for testing as the dataset doesn't have a separate test set
test_datagen = ImageDataGenerator(rescale=1./255)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary',
    shuffle=False
)

loss, accuracy = model.evaluate(test_generator)

print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")

Found 1966 images belonging to 2 classes.
62/62 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11s 185ms/step - accuracy: 0.9618 - loss: 0.1280
Test Loss: 0.1069
Test Accuracy: 0.9741
```

Kemudian, lakukan evaluasi model menggunakan data test agar model dapat memprediksi gambar selain yang kita berikan dari data *train* dan data *valid* seperti kode diatas. Lakukan evaluasi lebih lanjut sesuai kebutuhan.



```

from sklearn.metrics import classification_report
import numpy as np

true_classes = test_generator.classes
predictions = model.predict(test_generator)
predicted_classes = np.round(predictions).flatten()

class_labels = list(test_generator.class_indices.keys())

report = classification_report(true_classes, predicted_classes, target_names=class_labels)
print(report)

62/62 ━━━━━━━━━━ 12s 185ms/step
      precision    recall  f1-score   support

  not_pizza      0.98      0.97      0.97      983
    pizza         0.97      0.98      0.97      983

  accuracy           0.97      0.97      0.97     1966
  macro avg       0.97      0.97      0.97     1966
weighted avg       0.97      0.97      0.97     1966

```

## 8. Fungsi Klasifikasi Citra

Jika model sudah selesai dievaluasi menggunakan matriks, kita akan melakukan klasifikasi menggunakan model yang sudah kita latih. Gunakan kode berikut (penjelasan ada di gambar kode):

```

import numpy as np
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt

...
Tentukan parameter sesuai kebutuhan kalian,
target size disesuaikan dengan ukuran citra saat kalian melatih model
...

def classify_image(image_path, model, target_size=(150, 150)):
    img = image.load_img(image_path, target_size=target_size) #Memuat gambar dan mengubah ukuran sesuai dengan target size
    img_array = image.img_to_array(img) #Mengubah objek gambar menjadi array numpy agar sesuai dengan input yang dibutuhkan CNN
    img_array = np.expand_dims(img_array, axis=0) #Menambahkan dimensi baru di awal array karena model membutuhkan input dalam bentuk batch
    img_array /= 255.0 #Normalisasi nilai piksel pada gambar (sesuai dengan preprocessing yang dilakukan saat melatih model)

    prediction = model.predict(img_array)
    probability = prediction[0][0] #Membuat prediksi pada array citra yang sudah diproses

    #Mengambil nilai probabilitas dari hasil prediksi
    if probability > 0.5:
        predicted_class = 'pizza'
        confidence = probability * 100
    else:
        predicted_class = 'not pizza'
        confidence = (1 - probability) * 100

    plt.imshow(img)
    plt.title(f"Predicted: {predicted_class} ({confidence:.2f}%)")
    plt.axis('off')
    plt.show()

    return predicted_class

```



Sehingga, ketika fungsi diatas dipanggil, akan menghasilkan output sebagai berikut:

```
classify_image('/content/pizza_not_pizza/pizza/1008844.jpg', model)
1/1 ━━━━━━ 0s 130ms/step
Predicted: pizza (99.55%)

'pizza'

classify_image('/content/pizza_not_pizza/not_pizza/1043233.jpg', model)
1/1 ━━━━━━ 0s 57ms/step
Predicted: not pizza (94.28%)

'not pizza'
```

Jadi, ketika model sudah di-*training*, bukan berarti "muncul akurasi = selesai". Tapi, kita bisa melakukan percobaan dengan memberikan input citra di luar data *train* dan *valid* untuk menguji coba model apakah sesuai dengan hasil performa ketika dilakukan *training* dan evaluasi.

## LATIHAN PRAKTIKUM

---

Unduh dataset berikut: [LINK DATASET CODELAB](#)

Lengkapi dan perbaiki kode dibawah ini.

1. Load Dataset

```
Load Dataset
]
!unzip .... .zip
```



## 2. Preprocessing Data

```
from tensorflow.keras.preprocessing.image import ....  
  
dataset_path = '....'  
  
datagen = ImageDataGenerator(rescale=..., validation_split=...)  
  
train_generator = datagen.flow_from_directory(  
    dataset_path,  
    target_size=(...),  
    batch_size=...,  
    ...='binary',  
    subset='training'  
]  
validation_generator = datagen.flow_from_directory(  
    dataset_path,  
    target_size=(...),  
    batch_size=...,  
    ...='binary',  
    subset='validation'  
)  
  
test_generator = datagen.flow_from_directory[  
    dataset_path,  
    target_size=(...),  
    batch_size=...,  
    ...='binary',  
    subset='validation',  
    shuffle=...  
)
```

## 3. Build CNN Model

```
from tensorflow.keras.... import ...  
from tensorflow.keras.... import ...  
  
model = Sequential([  
    ...  
    ...  
)  
  
model.summary(fit())
```

#### 4. Compile Model

```
from tensorflow.keras.optimizers import ...

model.compile(optimizer=...,
              loss='...',
              metrics=['...'],
              callbacks=[...])
```

#### 5. Train Model

```
epoch = ...
history = model.compile(
    ...,
    epochs=...,
    validation_data=...,
    callbacks=[...]
    metrics=[...]
)
```

#### 6. Evaluate Model

```
loss, accuracy = model.evaluate(...)

print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")
```





## 7. Fungsi Klasifikasi Citra

```
import ... as np
from tensorflow.keras.preprocessing import ...
import ... as plt

def classify_new_image(...):
    img = ...
    img_array = image...(img)
    ...
    ...
    ...

    prediction = model.predict(img_array)
    probability = prediction[2][2]
    if ... > 0.5:
        predicted_class = 'car'
        confidence = probability * 100
    else:
        predicted_class = 'bike'
        confidence = (1 - ...) * 100

    plt.imshow(...)
    plt.title(f"Predicted: {...} ({confidence:.2f}%)")
    plt.axis('off')
    plt.show()

    return ...
```

```
image_to_classify = '...'
predicted_label = classify_new_image(...)
```

### RUBRIK PENILAIAN LATIHAN PRAKTIKUM

DETAIL	POIN	SELESAI
Model Berjalan Dengan Baik	50	<input type="checkbox"/>
Fungsi Klasifikasi Citra Berjalan Dengan Baik	50	<input type="checkbox"/>
<b>TOTAL</b>	<b>100</b>	



## TUGAS PRAKTIKUM

---

### LINK AKSES DATASET PRAKTIKUM

Tugas kalian adalah membangun model CNN untuk mengklasifikasikan gambar tersebut ke dalam salah satu dari dua kelas tersebut.

#### 1. **Eksplorasi Dataset**

- Muat dataset yang sudah diberikan, kemudian cek ukuran dataset
- Visualisasikan 5 contoh gambar dari masing-masing kelas

#### 2. **Preprocessing Data**

- Normalisasi nilai piksel citra
- Lakukan *preprocessing* sesuai kebutuhan **(DILARANG MELAKUKAN AUGMENTASI DATA)**

#### 3. **Membangun Model CNN**

- Buat model CNN sederhana (Convolution, Pooling, Fully Connected, dan output layer dengan aktivasi **sigmoid**).

#### 4. **Training Model**

- Gunakan loss function dan optimizer

#### 5. **Evaluasi Model**

- Uji model menggunakan data **test**
- Hitung metrik seperti accuracy, precision, recall, dan F1-score

#### 6. **Visualisasi Hasil**

- Tampilkan plot grafik *loss* dan *accuracy* untuk data *train* dan *valid*
- Tampilkan 5 gambar dari data *test* beserta prediksi model dan label aslinya

#### 7. **Fungsi Klasifikasi Citra**

- Buatlah fungsi model untuk menampilkan hasil prediksi. Berikan input citra secara live kepada asisten

Jelaskan performa model secara keseluruhan kepada asisten, serta berikan saran perbaikan untuk meningkatkan performa model. **Dilarang menggunakan (copy-paste) kode pada materi maupun codelab.**



## RUBRIK PENILAIAN PRAKTIKUM

DETAIL	POIN	SELESAI
Latihan Praktikum	10	<input type="checkbox"/>
Import Library dan Load Data	10	<input type="checkbox"/>
Preprocessing Data	10	<input type="checkbox"/>
Build CNN Model	10	<input type="checkbox"/>
Training Model	10	<input type="checkbox"/>
Evaluasi Model	10	<input type="checkbox"/>
Visualisasi dan Fungsi Hasil	20	<input type="checkbox"/>
Pemahaman Materi Berdasarkan Ketepatan Menjawab dan Penjelasan Program Kepada Asisten	20	<input type="checkbox"/>
<b>TOTAL</b>	<b>100</b>	

