Nama : Agilar Gumilar

Kls : TI3G

NIM : 2141720106

PRAKTIKUM 1

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
from zipfile import ZipFile
with ZipFile('/content/drive/MyDrive/ML2023/dataset.zip', 'r') as zipobj:
  zipobj.extractall('/content/drive/MyDrive/ML2023')
```

```
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
```

+ Kode + Teks

Langkah 2 - Pra Pengolahan Data

```
# Pra Pengolahan Data Training
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)
training_set = train_datagen.flow_from_directory('/content/drive/MyDrive/ML2023/dataset/training_set',
                                                 target_size = (64, 64),
                                                 batch_size = 32,
                                                 class_mode = 'binary')
```

```
Found 8017 images belonging to 2 classes.
```

```
# Pra Pengolahan Data Testing
test_datagen = ImageDataGenerator(rescale = 1./255)
test_set = test_datagen.flow_from_directory('/content/drive/MyDrive/ML2023/dataset/test_set',
                                            target_size = (64, 64),
                                            batch_size = 32,
                                            class_mode = 'binary')
```

```
Found 2000 images belonging to 2 classes.
```

Langkah 3 - Pembuatan Model CNN

```
# Langkah 3.1.  - Inisiasi Model CNN
cnn = tf.keras.models.Sequential()
```

```
# Langkah 3.2. - Pembuatan Layer Konvolusi 1
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu', input_shape=[64, 64, 3]))
```

```
# Langkah 3.3 - Pembuatan Layer Pooling 1
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

```
# Langkah 3.4 - Pembuatan Layer Konvolusi 2 dan Pooling 2
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

```
# Langkah 3.5 - Flattening
cnn.add(tf.keras.layers.Flatten())
```

```
# Langkah 3.6 - Fully Connected Layer 1 (Input)
cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))
```

```
# Langkah 3.7 - Fully Connected Layer 2 (Output)
cnn.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))


# Langkah 3.8 - Compile Model CNN
cnn.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Langkah 4 - Fit CNN

```
cnn.fit(x = training_set, validation_data = test_set, epochs = 25)
```

```
Epoch 1/25
251/251 [==============================] - 58s 185ms/step - loss: 0.6738 - accuracy: 0.5840 - val_loss: 0.6325 - val_accuracy: 0.6500
Epoch 2/25
251/251 [==============================] - 43s 171ms/step - loss: 0.6277 - accuracy: 0.6501 - val_loss: 0.5894 - val_accuracy: 0.7005
Epoch 3/25
251/251 [==============================] - 38s 153ms/step - loss: 0.5753 - accuracy: 0.6984 - val_loss: 0.5776 - val_accuracy: 0.7035
Epoch 4/25
251/251 [==============================] - 44s 174ms/step - loss: 0.5310 - accuracy: 0.7276 - val_loss: 0.5598 - val_accuracy: 0.7145
Epoch 5/25
251/251 [==============================] - 38s 152ms/step - loss: 0.5042 - accuracy: 0.7522 - val_loss: 0.5083 - val_accuracy: 0.7635
Epoch 6/25
251/251 [==============================] - 38s 152ms/step - loss: 0.4846 - accuracy: 0.7659 - val_loss: 0.5101 - val_accuracy: 0.7640
Epoch 7/25
251/251 [==============================] - 39s 155ms/step - loss: 0.4698 - accuracy: 0.7730 - val_loss: 0.4951 - val_accuracy: 0.7745
Epoch 8/25
251/251 [==============================] - 38s 151ms/step - loss: 0.4533 - accuracy: 0.7861 - val_loss: 0.4774 - val_accuracy: 0.7800
Epoch 9/25
251/251 [==============================] - 38s 150ms/step - loss: 0.4365 - accuracy: 0.7968 - val_loss: 0.4669 - val_accuracy: 0.7925
Epoch 10/25
251/251 [==============================] - 43s 173ms/step - loss: 0.4264 - accuracy: 0.8035 - val_loss: 0.4680 - val_accuracy: 0.7885
Epoch 11/25
251/251 [==============================] - 37s 149ms/step - loss: 0.4109 - accuracy: 0.8088 - val_loss: 0.4741 - val_accuracy: 0.7870
Epoch 12/25
251/251 [==============================] - 38s 153ms/step - loss: 0.4026 - accuracy: 0.8194 - val_loss: 0.4683 - val_accuracy: 0.7890
Epoch 13/25
251/251 [==============================] - 41s 163ms/step - loss: 0.3885 - accuracy: 0.8251 - val_loss: 0.4489 - val_accuracy: 0.8030
Epoch 14/25
251/251 [==============================] - 38s 151ms/step - loss: 0.3688 - accuracy: 0.8365 - val_loss: 0.4743 - val_accuracy: 0.7965
Epoch 15/25
251/251 [==============================] - 43s 172ms/step - loss: 0.3764 - accuracy: 0.8284 - val_loss: 0.4708 - val_accuracy: 0.7950
Epoch 16/25
251/251 [==============================] - 40s 158ms/step - loss: 0.3630 - accuracy: 0.8390 - val_loss: 0.4589 - val_accuracy: 0.7965
Epoch 17/25
251/251 [==============================] - 38s 150ms/step - loss: 0.3475 - accuracy: 0.8463 - val_loss: 0.5050 - val_accuracy: 0.7865
Epoch 18/25
251/251 [==============================] - 38s 150ms/step - loss: 0.3414 - accuracy: 0.8491 - val_loss: 0.4534 - val_accuracy: 0.8115
Epoch 19/25
251/251 [==============================] - 38s 150ms/step - loss: 0.3299 - accuracy: 0.8529 - val_loss: 0.4526 - val_accuracy: 0.8035
Epoch 20/25
251/251 [==============================] - 38s 150ms/step - loss: 0.3198 - accuracy: 0.8583 - val_loss: 0.4828 - val_accuracy: 0.8055
Epoch 21/25
251/251 [==============================] - 43s 171ms/step - loss: 0.3163 - accuracy: 0.8655 - val_loss: 0.4610 - val_accuracy: 0.8135
Epoch 22/25
251/251 [==============================] - 38s 151ms/step - loss: 0.2939 - accuracy: 0.8766 - val_loss: 0.5062 - val_accuracy: 0.7965
Epoch 23/25
251/251 [==============================] - 44s 174ms/step - loss: 0.2860 - accuracy: 0.8743 - val_loss: 0.4733 - val_accuracy: 0.8000
Epoch 24/25
251/251 [==============================] - 37s 148ms/step - loss: 0.2756 - accuracy: 0.8840 - val_loss: 0.4889 - val_accuracy: 0.8040
Epoch 25/25
251/251 [==============================] - 38s 151ms/step - loss: 0.2595 - accuracy: 0.8891 - val_loss: 0.5115 - val_accuracy: 0.7970
<keras.src.callbacks.History at 0x7f41e0f95f60>
```

Langkah 5 - Prediksi dengan 1 Citra

```
import numpy as np
from keras.preprocessing import image
test_image = image.load_img('/content/drive/MyDrive/ML2023/dataset/single_prediction/cat_or_dog_1.jpg', target_size = (64, 64))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = cnn.predict(test_image)
training_set.class_indices
if result[0][0] == 1:
  prediction = 'dog'
else:
  prediction = 'cat'
```

```
1/1 [==============================] - 0s 188ms/step
```

PRAKTIKUM 2

Langkah 1 - Load Library

```
import tensorflow as tf

from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

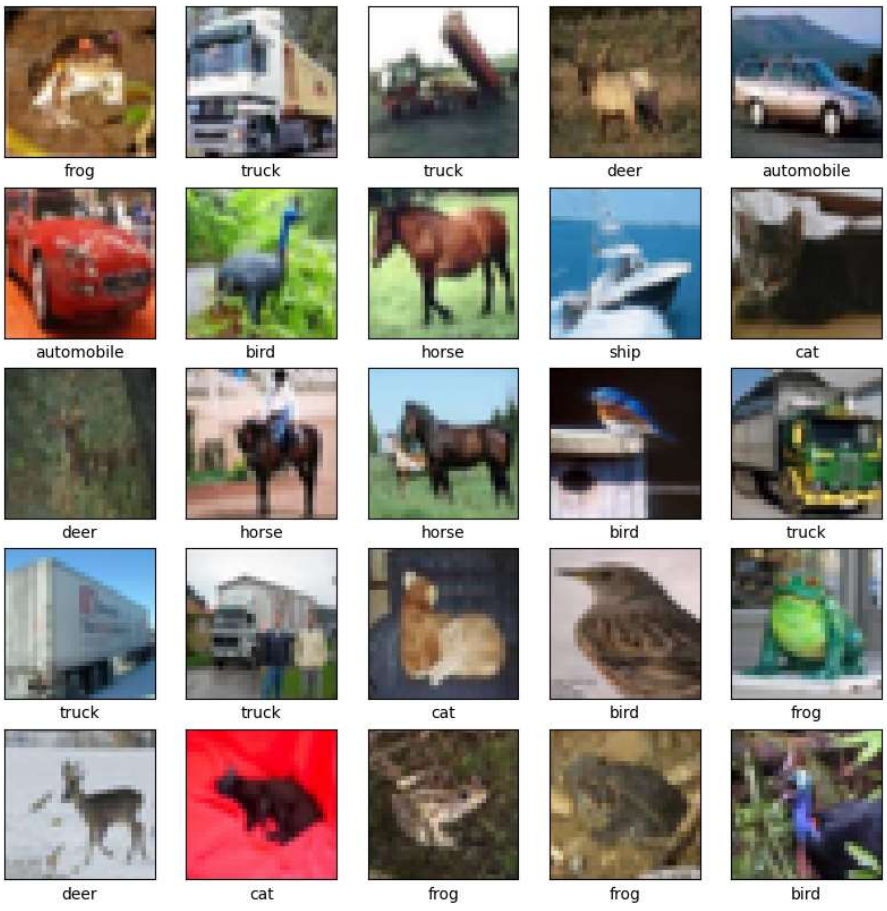
Langkah 2 - Unduh Dataset CIFAR

```
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 2s 0us/step
```

Langkah 3 - Verifikasi Data

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```



Langkah 4 - Buat Model CNN

```
# Langkah 4.1. - Buat Layer Konvolusi
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

```
# Langkah 4.2. - Cek Arsitektur Konvolusi
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 30, 30, 32)        896

 max_pooling2d_2 (MaxPoolin  (None, 15, 15, 32)        0
 g2D)

 conv2d_3 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_3 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 conv2d_4 (Conv2D)           (None, 4, 4, 64)          36928

=================================================================
Total params: 56320 (220.00 KB)
Trainable params: 56320 (220.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Langkah 4.3. - Tambahkan Layer Fully Connected
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
```

```
# Langkah 4.4. - Cek Arsitektur Model CNN
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 30, 30, 32)        896

 max_pooling2d_2 (MaxPoolin  (None, 15, 15, 32)        0
 g2D)

 conv2d_3 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_3 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 conv2d_4 (Conv2D)           (None, 4, 4, 64)          36928

 flatten_1 (Flatten)         (None, 1024)              0

 dense_2 (Dense)             (None, 64)                65600

 dense_3 (Dense)             (None, 10)                650

=================================================================
Total params: 122570 (478.79 KB)
Trainable params: 122570 (478.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Langkah 4.5. - Compile Model CNN
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Langkah 5 - Fit Model

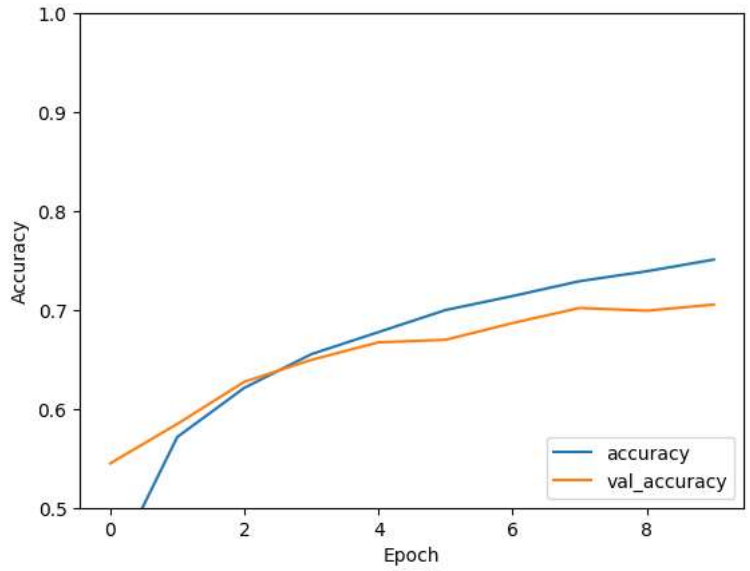
```
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

```
Epoch 1/10
1563/1563 [==============================] - 13s 7ms/step - loss: 1.5498 - accuracy: 0.4339 - val_loss: 1.2576 - val_accuracy: 0.5448
Epoch 2/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.2105 - accuracy: 0.5715 - val_loss: 1.1704 - val_accuracy: 0.5847
Epoch 3/10
1563/1563 [==============================] - 9s 6ms/step - loss: 1.0714 - accuracy: 0.6212 - val_loss: 1.0439 - val_accuracy: 0.6273
Epoch 4/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.9786 - accuracy: 0.6553 - val_loss: 0.9897 - val_accuracy: 0.6495
Epoch 5/10
1563/1563 [==============================] - 10s 6ms/step - loss: 0.9144 - accuracy: 0.6776 - val_loss: 0.9507 - val_accuracy: 0.6672
Epoch 6/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.8587 - accuracy: 0.6999 - val_loss: 0.9485 - val_accuracy: 0.6698
Epoch 7/10
1563/1563 [==============================] - 9s 6ms/step - loss: 0.8155 - accuracy: 0.7140 - val_loss: 0.9050 - val_accuracy: 0.6867
Epoch 8/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.7763 - accuracy: 0.7291 - val_loss: 0.8680 - val_accuracy: 0.7020
Epoch 9/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.7441 - accuracy: 0.7391 - val_loss: 0.8745 - val_accuracy: 0.6992
Epoch 10/10
1563/1563 [==============================] - 9s 5ms/step - loss: 0.7092 - accuracy: 0.7509 - val_loss: 0.8723 - val_accuracy: 0.7054
```

Langkah 6 - Evaluasi Model

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
```

```
313/313 - 1s - loss: 0.8723 - accuracy: 0.7054 - 1s/epoch - 5ms/step
```



Langkah 7 - Cetak Hasil Akurasi

```
print(test_acc)
```

```
0.7053999900817871
```

TUGAS

- Modifikasi model CNN pada praktikum 2 sehingga didapatkan akurasi testing lebih dari 80%.

Langkah 1 - Load Library

```
# Import library dan modul yang dibutuhkan
from tensorflow.keras.datasets import cifar10 #Mengimpor dataset CIFAR-10 yang disediakan oleh TensorFlow.
import numpy as np
import pandas as pd
import pickle
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import utils
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, Dropout, MaxPooling2D, GlobalAveragePooling2D, Activation, BatchNormalization
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import regularizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
from PIL import Image
```

Langkah 2 - Unduh Dataset CIFAR

```
# Memuat dataset CIFAR-10 dari TensorFlow
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# Menampilkan dimensi dataset
print('Training set shape:', X_train.shape)
print('Test set shape:', X_test.shape)
```

```
Training set shape: (50000, 32, 32, 3)
Test set shape: (10000, 32, 32, 3)
```

Langkah 3 - Normalisasi dan One-Hot Encoding

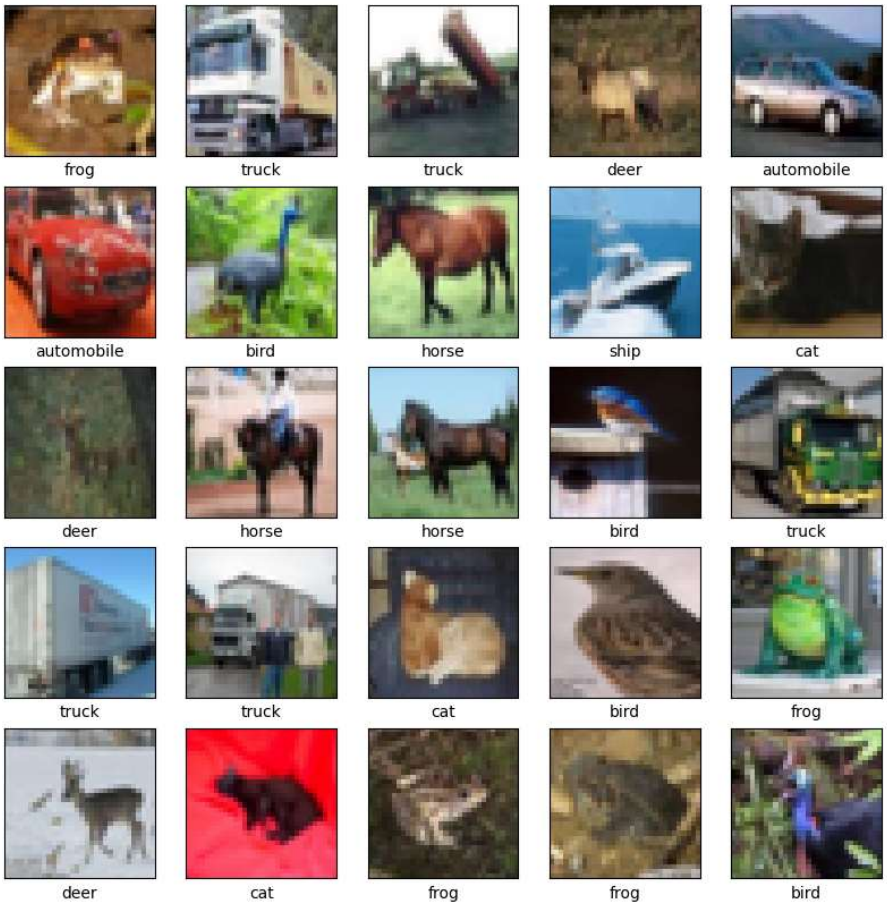
```
# Normalisasi nilai piksel ke dalam rentang 0-1
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0
num_classes = 10
# Mengubah label menjadi one-hot encoding
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

Langkah 4 - Split Data

```
# split data# Split data pelatihan menjadi data pelatihan dan data validasi
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.2, random_state=42)
```

Langkah 5 - Verifikasi Data

```
# Kelas-kelas untuk setiap label dalam dataset CIFAR-10
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

# Membuat plot dengan ukuran 10x10 inci
plt.figure(figsize=(10,10))

# Loop untuk menampilkan 25 gambar dan labelnya
for i in range(25):
   # Membuat subplot dengan ukuran 5x5 dan indeks i+1
   plt.subplot(5,5,i+1)

   # Menghilangkan label sumbu x dan y
   plt.xticks([])
   plt.yticks([])
   plt.grid(False) # Menonaktifkan grid
   plt.imshow(train_images[i]) # Menampilkan gambar pada indeks i dari data pelatihan
   # The CIFAR labels happen to be arrays,
   # which is why you need the extra index
   plt.xlabel(class_names[train_labels[i][0]])# Menambahkan label berdasarkan kelas gambar dari train_labels
plt.show() # Menampilkan plot
```



Langkah 6 - Buat Model CNN

```
def cnn_model():
    # Membuat model Sequential
    model = Sequential()
    # Layer konvolusi pertama dengan 128 filter, kernel size 3x3, fungsi aktivasi ReLU, padding same, regularisasi L2, dan input shape (32,
    model.add(Conv2D(filters=128, kernel_size=(3,3), activation='relu', padding='same', kernel_regularizer=regularizers.l2(1e-4), input_shap
    # Layer max pooling dengan pool size 2x2
    model.add(MaxPooling2D(pool_size=(2,2)))
    # Layer dropout untuk menghindari overfitting
    model.add(Dropout(0.3))
     # Layer konvolusi kedua dengan 256 filter, kernel size 3x3, fungsi aktivasi ReLU, padding same, regularisasi L2
    model.add(Conv2D(filters=256, kernel_size=(3,3), activation='relu', padding='same', kernel_regularizer=regularizers.l2(1e-4)))
    # Layer max pooling dengan pool size 2x2
    model.add(MaxPooling2D(pool_size=(2,2)))
    # Layer dropout
    model.add(Dropout(0.3))

    # Layer konvolusi ketiga dengan 512 filter, kernel size 3x3, fungsi aktivasi ReLU, padding same, regularisasi L2
    model.add(Conv2D(filters=512, kernel_size=(3,3), activation='relu', padding='same', kernel_regularizer=regularizers.l2(1e-4)))
    # Layer konvolusi keempat dengan 512 filter, kernel size 3x3, fungsi aktivasi ReLU, padding same, regularisasi L2
    model.add(Conv2D(filters=512, kernel_size=(3,3), activation='relu', padding='same', kernel_regularizer=regularizers.l2(1e-4)))
    # Layer konvolusi kelima dengan 256 filter, kernel size 3x3, fungsi aktivasi ReLU, padding same, regularisasi L2
    model.add(Conv2D(filters=256, kernel_size=(3,3), activation='relu', padding='same', kernel_regularizer=regularizers.l2(1e-4)))
    # Layer max pooling dengan pool size 2x2
    model.add(MaxPooling2D(pool_size=(2,2)))
    # Layer dropout
    model.add(Dropout(0.3))
    # Meratakan output menjadi vektor
    model.add(Flatten())

    # Fully connected layer pertama dengan 512 neuron dan fungsi aktivasi ReLU
    model.add(Dense(512, activation='relu'))
    # Layer dropout
    model.add(Dropout(0.5))
    # Fully connected layer kedua dengan 256 neuron dan fungsi aktivasi ReLU
    model.add(Dense(256, activation='relu'))
    # Layer dropout
    model.add(Dropout(0.5))
    # Fully connected layer ketiga dengan 128 neuron dan fungsi aktivasi ReLU
    model.add(Dense(128, activation='relu'))
    # Layer dropout
    model.add(Dropout(0.5))

    # Output layer dengan 10 neuron (sesuai jumlah kelas) dan fungsi aktivasi softmax
    model.add(Dense(10, activation='softmax'))
    # Menampilkan ringkasan arsitektur model
    model.summary()
    return model
```

Langkah 7 - Augmentasi Data

```
# Membuat objek ImageDataGenerator untuk augmentasi data
datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)
# Mengaplikasikan augmentasi data pada set pelatihan
datagen.fit(X_train)
```

Langkah 8 - Inisialisasi dan Kompilasi Model

```
# Memanggil fungsi cnn_model() untuk membuat model CNN
model = cnn_model()

# Mengompilasi model dengan fungsi loss, optimizer, dan metrik tertentu
model.compile(
    loss='categorical_crossentropy',  # Fungsi loss untuk klasifikasi multikelas
    optimizer=Adam(learning_rate=0.0003), # Pengoptimal Adam dengan learning rate 0.0003
    metrics=['accuracy']  # Metrik yang diukur adalah akurasi
)
```

```
Model: "sequential_2"
_________________________________________________________________
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 32, 32, 128)       3584

 max_pooling2d_4 (MaxPoolin  (None, 16, 16, 128)       0
 g2D)

 dropout (Dropout)           (None, 16, 16, 128)       0

 conv2d_6 (Conv2D)           (None, 16, 16, 256)       295168

 max_pooling2d_5 (MaxPoolin  (None, 8, 8, 256)         0
 g2D)

 dropout_1 (Dropout)         (None, 8, 8, 256)         0

 conv2d_7 (Conv2D)           (None, 8, 8, 512)         1180160

 conv2d_8 (Conv2D)           (None, 8, 8, 512)         2359808

 conv2d_9 (Conv2D)           (None, 8, 8, 256)         1179904

 max_pooling2d_6 (MaxPoolin  (None, 4, 4, 256)         0
 g2D)

 dropout_2 (Dropout)         (None, 4, 4, 256)         0

 flatten_2 (Flatten)         (None, 4096)              0

 dense_4 (Dense)             (None, 512)               2097664

 dropout_3 (Dropout)         (None, 512)               0

 dense_5 (Dense)             (None, 256)               131328

 dropout_4 (Dropout)         (None, 256)               0

 dense_6 (Dense)             (None, 128)               32896

 dropout_5 (Dropout)         (None, 128)               0

 dense_7 (Dense)             (None, 10)                1290

=================================================================
Total params: 7281802 (27.78 MB)
Trainable params: 7281802 (27.78 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Langkah 9 - Fit Model

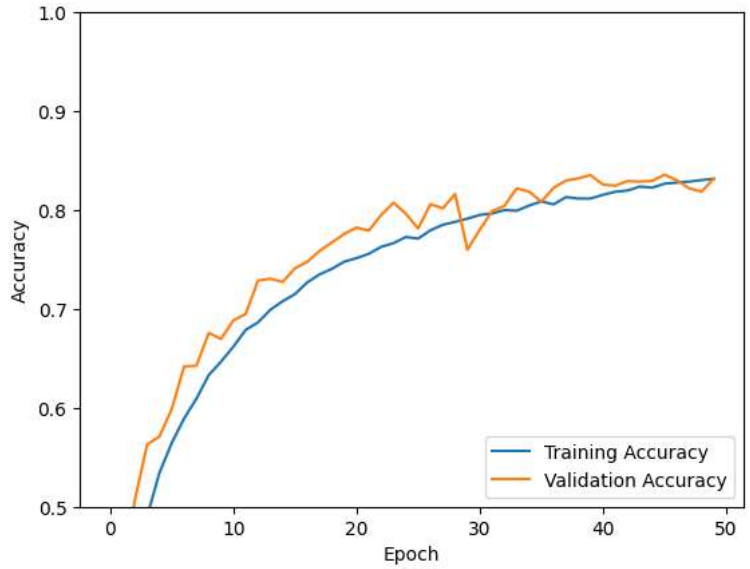
```
history = model.fit(
    datagen.flow(X_train, y_train, batch_size=64),
    steps_per_epoch=len(X_train) // 64,
    epochs=50,
    validation_data=(X_valid, y_valid),
    verbose=1
)
```

```
Epoch 1/50
625/625 [==============================] - 42s 55ms/step - loss: 2.2473 - accuracy: 0.1588 - val_loss: 1.8789 - val_accuracy: 0.2843
Epoch 2/50
625/625 [==============================] - 35s 56ms/step - loss: 1.8218 - accuracy: 0.3163 - val_loss: 1.5244 - val_accuracy: 0.4283
Epoch 3/50
625/625 [==============================] - 35s 56ms/step - loss: 1.6003 - accuracy: 0.4146 - val_loss: 1.3508 - val_accuracy: 0.5066
Epoch 4/50
625/625 [==============================] - 34s 54ms/step - loss: 1.4507 - accuracy: 0.4904 - val_loss: 1.2397 - val_accuracy: 0.5629
Epoch 5/50
625/625 [==============================] - 34s 54ms/step - loss: 1.3482 - accuracy: 0.5343 - val_loss: 1.1996 - val_accuracy: 0.5710
Epoch 6/50
625/625 [==============================] - 33s 52ms/step - loss: 1.2717 - accuracy: 0.5645 - val_loss: 1.1792 - val_accuracy: 0.5988
Epoch 7/50
625/625 [==============================] - 33s 52ms/step - loss: 1.2170 - accuracy: 0.5893 - val_loss: 1.0538 - val_accuracy: 0.6419
Epoch 8/50
625/625 [==============================] - 33s 52ms/step - loss: 1.1590 - accuracy: 0.6093 - val_loss: 1.0695 - val_accuracy: 0.6424
Epoch 9/50
625/625 [==============================] - 34s 54ms/step - loss: 1.1090 - accuracy: 0.6331 - val_loss: 0.9602 - val_accuracy: 0.6757
Epoch 10/50
625/625 [==============================] - 39s 62ms/step - loss: 1.0681 - accuracy: 0.6467 - val_loss: 0.9675 - val_accuracy: 0.6696
Epoch 11/50
625/625 [==============================] - 34s 55ms/step - loss: 1.0354 - accuracy: 0.6619 - val_loss: 0.9186 - val_accuracy: 0.6884
```

```
Epoch 12/50
625/625 [==============================] - 33s 53ms/step - loss: 0.9974 - accuracy: 0.6788 - val_loss: 0.9289 - val_accuracy: 0.6949
Epoch 13/50
625/625 [==============================] - 34s 55ms/step - loss: 0.9742 - accuracy: 0.6864 - val_loss: 0.8286 - val_accuracy: 0.7287
Epoch 14/50
625/625 [==============================] - 33s 53ms/step - loss: 0.9444 - accuracy: 0.6991 - val_loss: 0.8392 - val_accuracy: 0.7306
Epoch 15/50
625/625 [==============================] - 34s 55ms/step - loss: 0.9250 - accuracy: 0.7078 - val_loss: 0.8455 - val_accuracy: 0.7273
Epoch 16/50
625/625 [==============================] - 34s 54ms/step - loss: 0.9058 - accuracy: 0.7150 - val_loss: 0.8402 - val_accuracy: 0.7411
Epoch 17/50
625/625 [==============================] - 34s 55ms/step - loss: 0.8827 - accuracy: 0.7269 - val_loss: 0.7997 - val_accuracy: 0.7476
Epoch 18/50
625/625 [==============================] - 33s 53ms/step - loss: 0.8601 - accuracy: 0.7349 - val_loss: 0.7854 - val_accuracy: 0.7586
Epoch 19/50
625/625 [==============================] - 34s 54ms/step - loss: 0.8495 - accuracy: 0.7405 - val_loss: 0.7693 - val_accuracy: 0.7673
Epoch 20/50
625/625 [==============================] - 32s 51ms/step - loss: 0.8351 - accuracy: 0.7478 - val_loss: 0.7345 - val_accuracy: 0.7758
Epoch 21/50
625/625 [==============================] - 33s 52ms/step - loss: 0.8203 - accuracy: 0.7514 - val_loss: 0.7233 - val_accuracy: 0.7822
Epoch 22/50
625/625 [==============================] - 33s 53ms/step - loss: 0.8144 - accuracy: 0.7557 - val_loss: 0.7364 - val_accuracy: 0.7792
Epoch 23/50
625/625 [==============================] - 37s 60ms/step - loss: 0.7953 - accuracy: 0.7628 - val_loss: 0.6858 - val_accuracy: 0.7950
Epoch 24/50
625/625 [==============================] - 34s 54ms/step - loss: 0.7887 - accuracy: 0.7665 - val_loss: 0.6468 - val_accuracy: 0.8074
Epoch 25/50
625/625 [==============================] - 37s 60ms/step - loss: 0.7674 - accuracy: 0.7727 - val_loss: 0.6854 - val_accuracy: 0.7963
Epoch 26/50
625/625 [==============================] - 34s 54ms/step - loss: 0.7734 - accuracy: 0.7711 - val_loss: 0.7407 - val_accuracy: 0.7813
Epoch 27/50
625/625 [==============================] - 35s 55ms/step - loss: 0.7568 - accuracy: 0.7795 - val_loss: 0.6651 - val_accuracy: 0.8059
Epoch 28/50
625/625 [==============================] - 34s 54ms/step - loss: 0.7466 - accuracy: 0.7851 - val_loss: 0.6715 - val_accuracy: 0.8017
Epoch 29/50
```

Langkah 10 - Evaluasi Model

```
# Plot training accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1]) # Memastikan y-axis pada rentang 0.5 hingga 1
plt.legend(loc='lower right') # Menampilkan legenda pada posisi lower right
plt.show()

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
```




```
313/313 - 2s - loss: 0.6634 - accuracy: 0.8232 - 2s/epoch - 8ms/step
```

Langkah 11 - Cetak Hasil Akurasi

```
# Mencetak akurasi model pada data uji
print(test_acc)
```

```
0.823199987411499
```

- Buatlah model CNN untuk klasifikasi dataset MNIST

```
# Impor Library
# mengimpor TensorFlow untuk membuat dan melatih model neural network.
import tensorflow as tf
# Mengimpor modul layers dan models dari Keras, yang merupakan bagian dari TensorFlow untuk membangun model neural network.
from tensorflow.keras import layers, models
#  Mengimpor dataset MNIST yang berisi gambar digit tulisan tangan.
from tensorflow.keras.datasets import mnist
# Mengimpor fungsi to_categorical untuk melakukan one-hot encoding pada label.
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
```

```
# Memuat dan membagi dataset MNIST
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
# Normalisasi dan Reshape data
train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
# One-Hot Encoding label
train_labels = to_categorical(train_labels) # Melakukan one-hot encoding pada label pelatihan.
test_labels = to_categorical(test_labels) # Melakukan one-hot encoding pada label uji
```

```
# Membangun model CNN

#  Membuat model Sequential, yang berarti kita akan membangun model layer-by-layer secara berurutan.
model = models.Sequential()

# Menambahkan layer konvolusi dengan 32 filter, ukuran kernel 3x3, fungsi aktivasi ReLU, dan input_shape=(28, 28, 1) yang sesuai dengan dime
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))

# Menambahkan layer pooling maksimum dengan ukuran pool 2x2 untuk mengurangi dimensi gambar.
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
# Menambahkan layer flatten untuk meratakan output dari layer sebelumnya menjadi satu dimensi.
model.add(layers.Flatten())
# Menambahkan layer dense (fully connected) dengan 64 neuron dan fungsi aktivasi ReLU.
model.add(layers.Dense(64, activation='relu'))
# Menambahkan layer output dengan 10 neuron (sesuai jumlah kelas pada MNIST) dan fungsi aktivasi softmax untuk output klasifikasi multikelas
model.add(layers.Dense(10, activation='softmax'))
```

```
# Menampilkan ringkasan model
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_10 (Conv2D)          (None, 26, 26, 32)        320

 max_pooling2d_7 (MaxPoolin  (None, 13, 13, 32)        0
 g2D)

 conv2d_11 (Conv2D)          (None, 11, 11, 64)        18496

 max_pooling2d_8 (MaxPoolin  (None, 5, 5, 64)          0
 g2D)

 conv2d_12 (Conv2D)          (None, 3, 3, 64)          36928

 flatten_3 (Flatten)         (None, 576)               0

 dense_8 (Dense)             (None, 64)                36928

 dense_9 (Dense)             (None, 10)                650
```

```
=================================================================
Total params: 93322 (364.54 KB)
Trainable params: 93322 (364.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Mengompilasi model dengan menggunakan optimizer Adam
# fungsi loss categorical_crossentropy (karena ini adalah masalah klasifikasi multikelas), dan metrik akurasi.
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

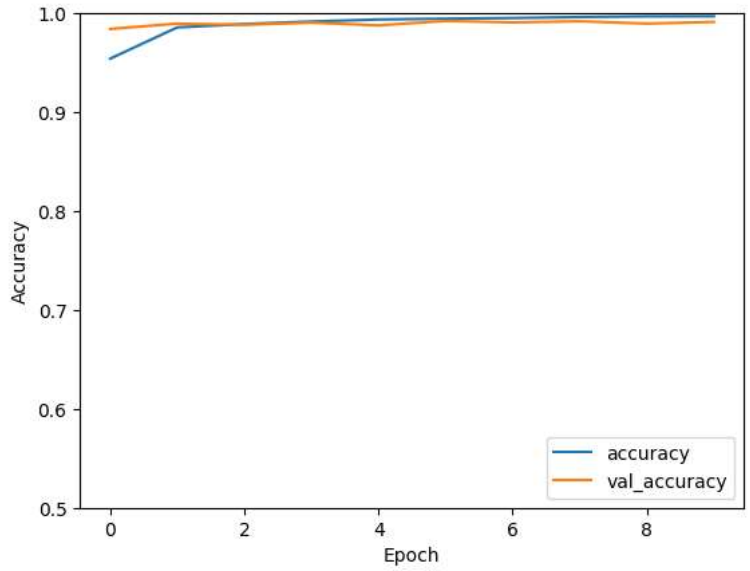
```
# Melatih model dengan data pelatihan selama 10 epoch, menggunakan data validasi untuk mengukur performa model selama pelatihan.
history = model.fit(train_images, train_labels, epochs=10, validation_data=(test_images, test_labels))
```

```
Epoch 1/10
1875/1875 [==============================] - 14s 6ms/step - loss: 0.1508 - accuracy: 0.9541 - val_loss: 0.0488 - val_accuracy: 0.9841
Epoch 2/10
1875/1875 [==============================] - 10s 6ms/step - loss: 0.0470 - accuracy: 0.9857 - val_loss: 0.0345 - val_accuracy: 0.9895
Epoch 3/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0344 - accuracy: 0.9890 - val_loss: 0.0372 - val_accuracy: 0.9883
Epoch 4/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0267 - accuracy: 0.9916 - val_loss: 0.0293 - val_accuracy: 0.9905
Epoch 5/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0195 - accuracy: 0.9937 - val_loss: 0.0461 - val_accuracy: 0.9877
Epoch 6/10
1875/1875 [==============================] - 10s 6ms/step - loss: 0.0172 - accuracy: 0.9945 - val_loss: 0.0273 - val_accuracy: 0.9922
Epoch 7/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0151 - accuracy: 0.9952 - val_loss: 0.0309 - val_accuracy: 0.9907
Epoch 8/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0117 - accuracy: 0.9962 - val_loss: 0.0322 - val_accuracy: 0.9919
Epoch 9/10
1875/1875 [==============================] - 12s 6ms/step - loss: 0.0094 - accuracy: 0.9968 - val_loss: 0.0423 - val_accuracy: 0.9895
Epoch 10/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0092 - accuracy: 0.9969 - val_loss: 0.0350 - val_accuracy: 0.9912
```

```
# Visualisasi akurasi pelatihan dan validasi
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
```

```
# Evaluasi model pada data uji
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
```

```
313/313 - 2s - loss: 0.0350 - accuracy: 0.9912 - 2s/epoch - 6ms/step
```




```
print(test_acc)
```

```
0.9911999702453613
```

```
# Fungsi untuk menampilkan contoh hasil prediksi
import numpy as np
import matplotlib.pyplot as plt

def visualize_predictions(images, true_labels, model, class_names, num_samples=25):
    # Menggunakan model untuk melakukan prediksi
    predictions = model.predict(images[:num_samples])

    # Mengubah prediksi menjadi indeks kelas dengan nilai tertinggi
    predicted_labels = np.argmax(predictions, axis=1)

    # Tampilkan hasil prediksi
    plt.figure(figsize=(12, 16))
    for i in range(num_samples):
        plt.subplot(5, 5, i+1)
        plt.xticks([])
        plt.yticks([])
        plt.grid(False)
        plt.imshow(images[i].reshape(28, 28), cmap=plt.cm.binary)

        true_label = class_names[true_labels[i]]
        predicted_label = class_names[predicted_labels[i]]
        color = 'green' if true_label == predicted_label else 'red'
        plt.xlabel(f'True: {true_label}\nPred: {predicted_label}', color=color)

    plt.show()

# Contoh definisi class_names
class_names = [str(i) for i in range(10)]

# Memanggil fungsi untuk menampilkan hasil prediksi pada set pelatihan
visualize_predictions(train_images, np.argmax(train_labels, axis=1), model, class_names)
```

```
1/1 [==============================] - 0s 189ms/step
```