A screen shot of a cell phone

AI-generated content may be incorrect.

3223600019

Muhammad Bimo Fachrizky

Membuat Aplikasi Deep Learning tensorflow 2

Senin, 05 Mei 2025

**Praktikum Sistem Cerdas**

**Praktikum 10**

Membuat Aplikasi Deep Learning tensorflow 2

1. Tujuan Pembelajaran

* Mahasiswa dapat memahami dan menjelaskan konsep Deep Learning
* Mahasiswa dapat menjelaskan model Deep Learning
* Mahasiswa dapat membuat aplikasi Deep Learing tensorflow

Software yang di perlukan

* Microsoft Visual C++
* PyCharm

1. Langkah percobaan
2. Listing 2.3. fashion\_classifier\_with\_callback.py

import tensorflow as tf

class MyCallback(tf.keras.callbacks.Callback):

def on\_epoch\_end(self, epoch, logs=None):

if logs.get("accuracy") > 0.6:

print("\nReached 60% accuracy so cancelling training!")

self.model.stop\_training = True

if \_\_name\_\_ == "\_\_main\_\_":

# Load the dataset

mnist = tf.keras.datasets.fashion\_mnist

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

# Normalize the data

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

# Initialize the callback

callbacks = MyCallback()

# Build the model

model = tf.keras.models.Sequential([

tf.keras.layers.Flatten(input\_shape=(28, 28)),

tf.keras.layers.Dense(512, activation=tf.nn.relu),

tf.keras.layers.Dense(10, activation=tf.nn.softmax)

])

# Compile the model

model.compile(

optimizer=tf.optimizers.Adam(),

loss="sparse\_categorical\_crossentropy",

metrics=["accuracy"]

)

# Train the model

model.fit(x\_train, y\_train, epochs=10, callbacks=[callbacks])

1. Listing 2.4. fashion\_classifier\_with\_cnn.py

import tensorflow as tf

import matplotlib.pyplot as plt

import numpy as np

def create\_cnn\_model():

mnist = tf.keras.datasets.fashion\_mnist

(training\_images, training\_labels), (test\_images, test\_labels) = mnist.load\_data()

training\_images = training\_images.reshape(-1, 28, 28, 1) / 255.0

test\_images = test\_images.reshape(-1, 28, 28, 1) / 255.0

model = tf.keras.models.Sequential([

tf.keras.layers.Conv2D(64, (3, 3), activation="relu", input\_shape=(28, 28, 1)),

tf.keras.layers.MaxPooling2D(2, 2),

tf.keras.layers.Conv2D(64, (3, 3), activation="relu"),

tf.keras.layers.MaxPooling2D(2, 2),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(128, activation="relu"),

tf.keras.layers.Dense(10, activation="softmax")

])

model.compile(

optimizer="adam",

loss="sparse\_categorical\_crossentropy",

metrics=["accuracy"]

)

model.summary()

model.fit(training\_images, training\_labels, epochs=10)

test\_loss = model.evaluate(test\_images, test\_labels)

return model, test\_loss

def visualizing\_conv\_and\_max\_pool(model):

mnist = tf.keras.datasets.fashion\_mnist

(\_, \_), (test\_images, test\_labels) = mnist.load\_data()

test\_images = test\_images.reshape(-1, 28, 28, 1) / 255.0

print(test\_labels[:100].reshape(10, 10))

f, ax\_arr = plt.subplots(3, 4, figsize=(10, 10))

first\_image = 0

second\_image = 28

third\_image = 23

convolution\_number = 2 # Nomor filter (0–63)

layer\_outputs = [layer.output for layer in model.layers]

activation\_model = tf.keras.models.Model(inputs=model.layers[0].input, outputs=layer\_outputs)

for x in range(4):

f1 = activation\_model.predict(test\_images[first\_image].reshape(1, 28, 28, 1))[x]

ax\_arr[0, x].imshow(f1[0, :, :, convolution\_number], cmap="inferno")

ax\_arr[0, x].grid(False)

f2 = activation\_model.predict(test\_images[second\_image].reshape(1, 28, 28, 1))[x]

ax\_arr[1, x].imshow(f2[0, :, :, convolution\_number], cmap="inferno")

ax\_arr[1, x].grid(False)

f3 = activation\_model.predict(test\_images[third\_image].reshape(1, 28, 28, 1))[x]

ax\_arr[2, x].imshow(f3[0, :, :, convolution\_number], cmap="inferno")

ax\_arr[2, x].grid(False)

plt.tight\_layout()

plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

cnn\_model, cnn\_test\_loss = create\_cnn\_model()

cnn\_model.save("model-saved/category2.h5")

visualizing\_conv\_and\_max\_pool(cnn\_model)

1. Listing 2.5. face\_expression\_classifier\_with\_cnn.py

import os

import zipfile

import numpy as np

import tensorflow as tf

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.preprocessing import image as keras\_image

class MyCallback(tf.keras.callbacks.Callback):

def \_\_init\_\_(self, desired\_accuracy):

super(MyCallback, self).\_\_init\_\_()

self.DESIRED\_ACCURACY = desired\_accuracy

def on\_epoch\_end(self, epoch, logs=None):

if logs is None:

logs = {}

if logs.get("accuracy") > self.DESIRED\_ACCURACY:

print(f"\nReached {self.DESIRED\_ACCURACY \* 100:.2f}% accuracy so cancelling training!")

self.model.stop\_training = True

def load\_dataset(zip\_file\_path, extracted\_zip\_file\_path, train\_happy\_dir, train\_sad\_dir):

if not os.path.exists(extracted\_zip\_file\_path):

os.makedirs(extracted\_zip\_file\_path, exist\_ok=True)

zip\_ref = zipfile.ZipFile(zip\_file\_path, "r")

zip\_ref.extractall(extracted\_zip\_file\_path)

zip\_ref.close()

train\_happy\_names = os.listdir(train\_happy\_dir)

train\_sad\_names = os.listdir(train\_sad\_dir)

print(train\_happy\_names[:10])

print(train\_sad\_names[:10])

print(f"Total training happy images: {len(train\_happy\_names)}")

print(f"Total training sad images: {len(train\_sad\_names)}")

return train\_happy\_names, train\_sad\_names

def do\_data\_preprocessing(dataset\_dir):

train\_datagen = ImageDataGenerator(rescale=1./255, validation\_split=0.2)

train\_generator = train\_datagen.flow\_from\_directory(

dataset\_dir,

target\_size=(150, 150),

batch\_size=8,

class\_mode="binary",

subset='training'

)

validation\_generator = train\_datagen.flow\_from\_directory(

dataset\_dir,

target\_size=(150, 150),

batch\_size=8,

class\_mode="binary",

subset='validation'

)

return train\_generator, validation\_generator

def create\_cnn\_model():

model = tf.keras.models.Sequential([

tf.keras.layers.Conv2D(16, (3, 3), activation="relu", input\_shape=(150, 150, 3)),

tf.keras.layers.MaxPooling2D(2, 2),

tf.keras.layers.Conv2D(32, (3, 3), activation="relu"),

tf.keras.layers.MaxPooling2D(2, 2),

tf.keras.layers.Conv2D(64, (3, 3), activation="relu"),

tf.keras.layers.MaxPooling2D(2, 2),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(512, activation="relu"),

tf.keras.layers.Dense(1, activation="sigmoid")

])

model.compile(

loss="binary\_crossentropy",

optimizer=Adam(learning\_rate=0.001),

metrics=["accuracy"]

)

model.summary()

return model

def plot\_training\_images(train\_happy\_dir, train\_sad\_dir, train\_happy\_names, train\_sad\_names):

plt.figure(figsize=(16, 16))

img\_index = 0

next\_happy\_img = [os.path.join(train\_happy\_dir, fname) for fname in train\_happy\_names[img\_index:img\_index + 8]]

next\_sad\_img = [os.path.join(train\_sad\_dir, fname) for fname in train\_sad\_names[img\_index:img\_index + 8]]

for i, img\_path in enumerate(next\_happy\_img + next\_sad\_img):

plt.subplot(4, 4, i + 1)

plt.axis("off")

img = mpimg.imread(img\_path)

plt.imshow(img)

plt.tight\_layout()

plt.show()

def classify\_images(fn\_arr, model):

for fn in fn\_arr:

path = os.path.join("datasets", fn)

if not os.path.exists(path):

print(f"File not found: {path}")

continue

img = keras\_image.load\_img(path, target\_size=(150, 150))

x = keras\_image.img\_to\_array(img)

x = np.expand\_dims(x, axis=0)

x /= 255.0 # Normalize the image

prediction = model.predict(x)

print(f"Prediction score for {fn}: {prediction[0][0]}")

if prediction[0] > 0.5:

print(f"{fn} is happy")

else:

print(f"{fn} is sad")

if \_\_name\_\_ == "\_\_main\_\_":

# Konfigurasi path

base\_dir = "datasets"

zip\_file\_path = os.path.join(base\_dir, "happy-or-sad.zip")

extracted\_zip\_file\_path = os.path.join(base\_dir, "happy-or-sad")

train\_happy\_dir = os.path.join(extracted\_zip\_file\_path, "happy")

train\_sad\_dir = os.path.join(extracted\_zip\_file\_path, "sad")

# Load dataset

train\_happy\_names, train\_sad\_names = load\_dataset(

zip\_file\_path, extracted\_zip\_file\_path,

train\_happy\_dir, train\_sad\_dir

)

plot\_training\_images(train\_happy\_dir, train\_sad\_dir, train\_happy\_names, train\_sad\_names)

train\_generator, validation\_generator = do\_data\_preprocessing(extracted\_zip\_file\_path)

cnn\_model = create\_cnn\_model()

DESIRED\_ACCURACY = 0.99

callbacks = MyCallback(DESIRED\_ACCURACY)

history = cnn\_model.fit(

train\_generator,

steps\_per\_epoch=train\_generator.samples // train\_generator.batch\_size,

validation\_data=validation\_generator,

validation\_steps=validation\_generator.samples // validation\_generator.batch\_size,

epochs=50,

verbose=1,

callbacks=[callbacks]

)

test\_images = [

"beauty-1132617\_640.jpg",

"girl-2961959\_640.jpg",

"woman-2126727\_640.jpg",

"beautiful-18279\_640.jpg"

]

classify\_images(test\_images, cnn\_model)

1. Listing 3.1. cats\_and\_dogs\_classifier\_with\_cnn.py

import os

import zipfile

import numpy as np

import random

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

import tensorflow as tf

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.preprocessing.image import ImageDataGenerator, img\_to\_array, load\_img

from tensorflow.keras.preprocessing import image as keras\_image

def load\_dataset(zip\_file\_path, extracted\_zip\_file\_path):

zip\_ref = zipfile.ZipFile(zip\_file\_path, "r")

zip\_ref.extractall(os.path.split(extracted\_zip\_file\_path)[0])

zip\_ref.close()

train\_dir = os.path.join(extracted\_zip\_file\_path, "train")

validation\_dir = os.path.join(extracted\_zip\_file\_path, "validation")

train\_cats\_dir = os.path.join(train\_dir, "cats")

train\_dogs\_dir = os.path.join(train\_dir, "dogs")

validation\_cats\_dir = os.path.join(validation\_dir, "cats")

validation\_dogs\_dir = os.path.join(validation\_dir, "dogs")

train\_cat\_fnames = os.listdir(train\_cats\_dir)

train\_dog\_fnames = os.listdir(train\_dogs\_dir)

validation\_cat\_fnames = os.listdir(validation\_cats\_dir)

validation\_dog\_fnames = os.listdir(validation\_dogs\_dir)

print(train\_cat\_fnames[:10])

print(train\_dog\_fnames[:10])

print("total training cat images :", len(train\_cat\_fnames))

print("total training dog images :", len(train\_dog\_fnames))

print("total validation cat images:", len(validation\_cat\_fnames))

print("total validation dog images:", len(validation\_dog\_fnames))

return train\_dir, validation\_dir, train\_cats\_dir, train\_dogs\_dir, train\_cat\_fnames, train\_dog\_fnames

def do\_data\_preprocessing(train\_dir, validation\_dir):

train\_datagen = ImageDataGenerator(rescale=1. / 255)

validation\_datagen = ImageDataGenerator(rescale=1. / 255)

train\_generator = train\_datagen.flow\_from\_directory(

train\_dir,

batch\_size=20,

class\_mode="binary",

target\_size=(150, 150)

)

validation\_generator = validation\_datagen.flow\_from\_directory(

validation\_dir,

batch\_size=20,

class\_mode="binary",

target\_size=(150, 150)

)

return train\_generator, validation\_generator

def create\_cnn\_model():

inputs = tf.keras.Input(shape=(150, 150, 3))

x = tf.keras.layers.Conv2D(16, (3, 3), activation="relu")(inputs)

x = tf.keras.layers.MaxPooling2D(2, 2)(x)

x = tf.keras.layers.Conv2D(32, (3, 3), activation="relu")(x)

x = tf.keras.layers.MaxPooling2D(2, 2)(x)

x = tf.keras.layers.Conv2D(64, (3, 3), activation="relu")(x)

x = tf.keras.layers.MaxPooling2D(2, 2)(x)

x = tf.keras.layers.Flatten()(x)

x = tf.keras.layers.Dense(512, activation="relu")(x)

outputs = tf.keras.layers.Dense(1, activation="sigmoid")(x)

model = tf.keras.Model(inputs=inputs, outputs=outputs)

model.summary()

model.compile(optimizer=Adam(learning\_rate=0.001),

loss="binary\_crossentropy",

metrics=["accuracy"])

return model

def plot\_cats\_and\_dogs(train\_cats\_dir, train\_dogs\_dir, train\_cat\_fnames, train\_dog\_fnames):

nrows, ncols = 4, 4

fig = plt.gcf()

fig.set\_size\_inches(ncols \* 4, nrows \* 4)

pic\_index = 8

next\_cat\_pix = [os.path.join(train\_cats\_dir, fname) for fname in train\_cat\_fnames[pic\_index - 8:pic\_index]]

next\_dog\_pix = [os.path.join(train\_dogs\_dir, fname) for fname in train\_dog\_fnames[pic\_index - 8:pic\_index]]

for i, img\_path in enumerate(next\_cat\_pix + next\_dog\_pix):

sp = plt.subplot(nrows, ncols, i + 1)

sp.axis("off")

img = mpimg.imread(img\_path)

plt.imshow(img)

plt.pause(0)

def classify\_images(fn\_arr, model):

for fn in fn\_arr:

path = os.path.join("datasets", fn)

img = keras\_image.load\_img(path, target\_size=(150, 150))

x = keras\_image.img\_to\_array(img)

x = np.expand\_dims(x, axis=0)

image\_i = np.vstack([x])

classes = model.predict(image\_i, batch\_size=10)

print(classes[0])

if classes[0] > 0.5:

print(fn + " is a dog")

else:

print(fn + " is a cat")

def plot\_intermediate\_repr(model, train\_cats\_dir, train\_dogs\_dir, train\_cat\_fnames, train\_dog\_fnames):

successive\_outputs = [layer.output for layer in model.layers]

visualization\_model = tf.keras.models.Model(inputs=model.input, outputs=successive\_outputs)

cat\_img\_files = [os.path.join(train\_cats\_dir, f) for f in train\_cat\_fnames]

dog\_img\_files = [os.path.join(train\_dogs\_dir, f) for f in train\_dog\_fnames]

img\_path = random.choice(cat\_img\_files + dog\_img\_files)

img = load\_img(img\_path, target\_size=(150, 150))

x = img\_to\_array(img)

x = x.reshape((1,) + x.shape) / 255.

successive\_feature\_maps = visualization\_model.predict(x)

layer\_names = [layer.name for layer in model.layers]

for layer\_name, feature\_map in zip(layer\_names, successive\_feature\_maps):

if len(feature\_map.shape) == 4:

n\_features = feature\_map.shape[-1]

size = feature\_map.shape[1]

display\_grid = np.zeros((size, size \* n\_features))

for i in range(n\_features):

x = feature\_map[0, :, :, i]

x -= x.mean()

x = x / x.std() if x.std() > 1e-14 else x

x = np.clip(x \* 64 + 128, 0, 255).astype("uint8")

display\_grid[:, i \* size:(i + 1) \* size] = x

plt.figure(figsize=(20, 2))

plt.title(layer\_name)

plt.grid(False)

plt.imshow(display\_grid, aspect="auto", cmap="viridis")

plt.subplots\_adjust(left=0.03, right=0.99)

plt.pause(0)

def plot\_history(train, val, title):

epochs = range(len(train))

plt.figure()

plt.plot(epochs, train, label="train")

plt.plot(epochs, val, label="val")

plt.title(title)

plt.legend(loc="best")

plt.pause(0)

if \_\_name\_\_ == "\_\_main\_\_":

zip\_file\_path = "datasets/cats\_and\_dogs\_filtered.zip"

extracted\_zip\_file\_path = "datasets/cats\_and\_dogs\_filtered"

train\_dir, validation\_dir, train\_cats\_dir, train\_dogs\_dir, train\_cat\_fnames, train\_dog\_fnames = load\_dataset(

zip\_file\_path, extracted\_zip\_file\_path)

plot\_cats\_and\_dogs(train\_cats\_dir, train\_dogs\_dir, train\_cat\_fnames, train\_dog\_fnames)

train\_generator, validation\_generator = do\_data\_preprocessing(train\_dir, validation\_dir)

cnn\_model = create\_cnn\_model()

history = cnn\_model.fit(

train\_generator,

validation\_data=validation\_generator,

steps\_per\_epoch=100,

epochs=15,

validation\_steps=50,

verbose=1

)

fn\_arr = [

"cat-2083492\_only\_head.jpg",

"cat-1146504\_640.jpg",

"dog-3846767\_640.jpg",

"dog-3388069\_640.jpg"

]

classify\_images(fn\_arr, cnn\_model)

plot\_intermediate\_repr(cnn\_model, train\_cats\_dir, train\_dogs\_dir, train\_cat\_fnames, train\_dog\_fnames)

acc = history.history["accuracy"]

val\_acc = history.history["val\_accuracy"]

loss = history.history["loss"]

val\_loss = history.history["val\_loss"]

plot\_history(acc, val\_acc, "Training and validation accuracy")

plot\_history(loss, val\_loss, "Training and validation loss")

1. Listing 3.2. cats\_and\_dogs\_classifier\_with\_imagedatagen\_and\_dropout.py

import os

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras import layers

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.preprocessing import image as keras\_image

# Constants

IMG\_HEIGHT = 150

IMG\_WIDTH = 150

BATCH\_SIZE = 20

# Plot training history

def plot\_history(train, val, title):

epochs = range(len(train))

plt.figure()

plt.plot(epochs, train, label="train")

plt.plot(epochs, val, label="val")

plt.title(title)

plt.legend(loc="best")

plt.show()

# Create CNN model

def create\_cnn\_model(input\_shape=(150, 150, 3)):

model = tf.keras.models.Sequential([

tf.keras.layers.Input(shape=input\_shape),

tf.keras.layers.Conv2D(32, (3, 3), activation="relu"),

tf.keras.layers.MaxPooling2D(2, 2),

tf.keras.layers.Conv2D(64, (3, 3), activation="relu"),

tf.keras.layers.MaxPooling2D(2, 2),

tf.keras.layers.Conv2D(128, (3, 3), activation="relu"),

tf.keras.layers.MaxPooling2D(2, 2),

tf.keras.layers.Conv2D(128, (3, 3), activation="relu"),

tf.keras.layers.MaxPooling2D(2, 2),

tf.keras.layers.Dropout(0.5),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(512, activation="relu"),

tf.keras.layers.Dense(1, activation="sigmoid")

])

model.compile(

loss="binary\_crossentropy",

optimizer=Adam(learning\_rate=1e-4),

metrics=["accuracy"]

)

model.summary()

return model

# Classify individual images

def classify\_images(fn\_arr, model):

for fn in fn\_arr:

path = os.path.join("datasets", fn)

if not os.path.exists(path):

print(f"Warning: File {path} not found.")

continue

img = keras\_image.load\_img(path, target\_size=(150, 150))

x = keras\_image.img\_to\_array(img)

x = np.expand\_dims(x, axis=0) / 255.0

prediction = model.predict(x)

print(f"File: {fn}, Prediction score: {prediction[0][0]}")

if prediction[0][0] > 0.5:

print(f"{fn} is a dog")

else:

print(f"{fn} is a cat")

# Build augmentation pipeline

def build\_augmentation\_pipeline(img\_height, img\_width):

data\_augmentation = tf.keras.Sequential([

layers.RandomFlip("horizontal", input\_shape=(img\_height, img\_width, 3)),

layers.RandomRotation(factor=0.11, fill\_mode="nearest"),

layers.RandomZoom(height\_factor=0.2, width\_factor=0.2, fill\_mode="nearest"),

layers.RandomTranslation(height\_factor=0.2, width\_factor=0.2, fill\_mode="nearest")

])

return data\_augmentation

# Image preprocessing

def process\_path(file\_path):

img = tf.io.read\_file(file\_path)

img = tf.io.decode\_image(img, channels=3, expand\_animations=False)

img = tf.image.resize(img, [IMG\_HEIGHT, IMG\_WIDTH])

return img

# Dataset configuration

def configure\_for\_performance(ds, shuffle=False, augment=False):

ds = ds.cache()

if shuffle:

ds = ds.shuffle(buffer\_size=1000)

ds = ds.batch(BATCH\_SIZE)

if augment:

augmentation\_pipeline = build\_augmentation\_pipeline(IMG\_HEIGHT, IMG\_WIDTH)

ds = ds.map(lambda x, y: (augmentation\_pipeline(x, training=True), y),

num\_parallel\_calls=tf.data.AUTOTUNE)

ds = ds.map(lambda x, y: (tf.cast(x, tf.float32) / 255.0, tf.cast(y, tf.float32)),

num\_parallel\_calls=tf.data.AUTOTUNE)

ds = ds.prefetch(buffer\_size=tf.data.AUTOTUNE)

return ds

# Load and prepare datasets

def do\_data\_preprocessing\_tfdata(train\_dir, validation\_dir, aug=False):

train\_ds = tf.keras.utils.image\_dataset\_from\_directory(

train\_dir,

labels='inferred',

label\_mode='binary',

image\_size=(IMG\_HEIGHT, IMG\_WIDTH),

interpolation='nearest',

batch\_size=None,

shuffle=True,

seed=123

)

val\_ds = tf.keras.utils.image\_dataset\_from\_directory(

validation\_dir,

labels='inferred',

label\_mode='binary',

image\_size=(IMG\_HEIGHT, IMG\_WIDTH),

interpolation='nearest',

batch\_size=None,

shuffle=False

)

class\_names = train\_ds.class\_names

print("Class names:", class\_names)

train\_ds = configure\_for\_performance(train\_ds, shuffle=True, augment=aug)

val\_ds = configure\_for\_performance(val\_ds, shuffle=False, augment=False)

return train\_ds, val\_ds

# Main execution block

if \_\_name\_\_ == "\_\_main\_\_":

base\_dir = "datasets/cats\_and\_dogs\_filtered"

train\_dir = os.path.join(base\_dir, "train")

validation\_dir = os.path.join(base\_dir, "validation")

train\_cats\_dir = os.path.join(train\_dir, "cats")

train\_dogs\_dir = os.path.join(train\_dir, "dogs")

validation\_cats\_dir = os.path.join(validation\_dir, "cats")

validation\_dogs\_dir = os.path.join(validation\_dir, "dogs")

if not os.path.exists(train\_dir) or not os.path.exists(validation\_dir):

print(f"Error: Dataset directory {base\_dir} or subdirectories not found.")

exit()

if not all(os.path.exists(d) for d in [train\_cats\_dir, train\_dogs\_dir, validation\_cats\_dir, validation\_dogs\_dir]):

print("Error: One or more subdirectories (cats/dogs) for train/validation not found.")

exit()

try:

print(f"Training cat images: {len(os.listdir(train\_cats\_dir))}")

print(f"Training dog images: {len(os.listdir(train\_dogs\_dir))}")

print(f"Validation cat images: {len(os.listdir(validation\_cats\_dir))}")

print(f"Validation dog images: {len(os.listdir(validation\_dogs\_dir))}")

train\_generator, validation\_generator = do\_data\_preprocessing\_tfdata(

train\_dir, validation\_dir, aug=True

)

cnn\_model = create\_cnn\_model(input\_shape=(IMG\_HEIGHT, IMG\_WIDTH, 3))

history = cnn\_model.fit(

train\_generator,

epochs=100,

validation\_data=validation\_generator,

verbose=1

)

acc = history.history["accuracy"]

val\_acc = history.history["val\_accuracy"]

loss = history.history["loss"]

val\_loss = history.history["val\_loss"]

plot\_history(acc, val\_acc, "Training and validation accuracy")

plot\_history(loss, val\_loss, "Training and validation loss")

test\_images = [

"cat-2083492\_only\_head.jpg",

"cat-1146504\_640.jpg",

"dog-3846767\_640.jpg",

"dog-3388069\_640.jpg"

]

classify\_images(test\_images, cnn\_model)

except FileNotFoundError as e:

print(f"FileNotFoundError occurred: {e}")

except Exception as e:

print(f"An error occurred: {e}")

1. Listing 3.3. cats\_and\_dogs\_classifier\_with\_transfer\_learning.py

import os

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras import layers, Model

from tensorflow.keras.applications.vgg16 import VGG16

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.preprocessing import image as keras\_image

def plot\_history(train, val, title):

epochs = range(len(train))

plt.figure()

plt.plot(epochs, train, label="train")

plt.plot(epochs, val, label="val")

plt.title(title)

plt.legend(loc="best")

plt.pause(1.0)

def do\_data\_preprocessing(train\_dir, validation\_dir, aug=False):

if aug:

train\_datagen = ImageDataGenerator(

rescale=1./255,

rotation\_range=40,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

shear\_range=0.2,

zoom\_range=0.2,

horizontal\_flip=True,

fill\_mode="nearest"

)

else:

train\_datagen = ImageDataGenerator(rescale=1./255)

validation\_datagen = ImageDataGenerator(rescale=1./255)

train\_generator = train\_datagen.flow\_from\_directory(

train\_dir,

batch\_size=20,

class\_mode="binary",

target\_size=(150, 150)

)

validation\_generator = validation\_datagen.flow\_from\_directory(

validation\_dir,

batch\_size=20,

class\_mode="binary",

target\_size=(150, 150)

)

return train\_generator, validation\_generator

def create\_cnn\_model(local\_weights\_file):

pre\_trained\_model = VGG16(

input\_shape=(150, 150, 3),

include\_top=False,

weights=None

)

pre\_trained\_model.load\_weights(local\_weights\_file)

for layer in pre\_trained\_model.layers:

layer.trainable = False

pre\_trained\_model.summary()

last\_layer = pre\_trained\_model.get\_layer("block5\_pool")

print("last layer output shape:", last\_layer.output.shape)

last\_output = last\_layer.output

x = layers.Dropout(0.2)(last\_output)

x = layers.Flatten()(x)

x = layers.Dense(1024, activation="relu")(x)

x = layers.Dense(1, activation="sigmoid")(x)

model = Model(pre\_trained\_model.input, x)

model.compile(

optimizer=Adam(learning\_rate=1e-4),

loss="binary\_crossentropy",

metrics=["accuracy"]

)

return model

def classify\_images(fn\_arr, model):

for fn in fn\_arr:

path = os.path.join("datasets", fn)

if not os.path.exists(path):

print(f"File not found: {path}")

continue

img = keras\_image.load\_img(path, target\_size=(150, 150))

x = keras\_image.img\_to\_array(img)

x = np.expand\_dims(x, axis=0)

image\_i = np.vstack([x])

classes = model.predict(image\_i, batch\_size=10)

print(f"{fn} prediction score: {classes[0][0]}")

if classes[0] > 0.5:

print(fn + " is a dog")

else:

print(fn + " is a cat")

if \_\_name\_\_ == "\_\_main\_\_":

local\_weights\_file = "pre-trained/vgg16\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5"

base\_dir = "datasets/cats\_and\_dogs\_filtered"

train\_dir = os.path.join(base\_dir, "train")

validation\_dir = os.path.join(base\_dir, "validation")

train\_cats\_dir = os.path.join(train\_dir, "cats")

train\_dogs\_dir = os.path.join(train\_dir, "dogs")

validation\_cats\_dir = os.path.join(validation\_dir, "cats")

validation\_dogs\_dir = os.path.join(validation\_dir, "dogs")

train\_cat\_fnames = os.listdir(train\_cats\_dir)

train\_dog\_fnames = os.listdir(train\_dogs\_dir)

print(f"Number of cat training images: {len(train\_cat\_fnames)}")

print(f"Number of dog training images: {len(train\_dog\_fnames)}")

train\_generator, validation\_generator = do\_data\_preprocessing(

train\_dir, validation\_dir, aug=True

)

cnn\_model = create\_cnn\_model(local\_weights\_file)

history = cnn\_model.fit(

train\_generator,

epochs=20,

validation\_data=validation\_generator,

verbose=1

)

acc = history.history["accuracy"]

val\_acc = history.history["val\_accuracy"]

loss = history.history["loss"]

val\_loss = history.history["val\_loss"]

plot\_history(acc, val\_acc, "Training and validation accuracy")

plot\_history(loss, val\_loss, "Training and validation loss")

fn\_arr = [

"cat-2083492\_only\_head.jpg",

"cat-1146504\_640.jpg",

"dog-3846767\_640.jpg",

"dog-3388069\_640.jpg"

]

classify\_images(fn\_arr, cnn\_model)

1. Hasil Listing 3.4. hand\_sign\_language\_classifier.py

import os

import sys

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.preprocessing import image as keras\_image

import tensorflow as tf

def get\_data(filename):

with open(filename) as training\_file:

\_ = training\_file.readline() # skip first line

data = training\_file.readlines()

labels = []

images = []

num\_of\_data = len(data)

for i, row in enumerate(data):

row = row.strip("\n").split(",")

labels.append(row[0])

images.append(np.array\_split(row[1:785], 28))

sys.stdout.write(f"\rprocessing: {(i + 1) / float(num\_of\_data) \* 100:.2f} %")

sys.stdout.flush()

print("")

labels = np.array(labels).astype(float)

images = np.array(images).astype(float)

return images, labels

def plot\_one\_image(image\_data, image\_label):

fig, ax = plt.subplots()

ax.imshow(image\_data, cmap="gray", vmin=0, vmax=255)

num\_to\_alphabet = [

'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',

'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S',

'T', 'U', 'V', 'W', 'X', 'Y'

]

ax.set\_title("image\_label = {:g} ({:s})".format(image\_label,

num\_to\_alphabet[int(image\_label)]))

plt.show()

plt.pause(1.0)

def do\_data\_preprocessing(training\_images, training\_labels,

validation\_images, validation\_labels):

train\_datagen = ImageDataGenerator(

rescale=1. / 255,

rotation\_range=40,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

shear\_range=0.2,

zoom\_range=0.2,

fill\_mode="nearest"

)

validation\_datagen = ImageDataGenerator(rescale=1. / 255)

training\_generator = train\_datagen.flow(

training\_images,

training\_labels,

batch\_size=20,

)

validation\_generator = validation\_datagen.flow(

validation\_images,

validation\_labels,

batch\_size=20

)

return training\_generator, validation\_generator

def create\_cnn\_model():

model = tf.keras.models.Sequential([

tf.keras.layers.Conv2D(64, (3, 3), activation="relu", input\_shape=(28, 28, 1)),

tf.keras.layers.MaxPooling2D(2, 2),

tf.keras.layers.Dropout(0.1),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(1024, activation="relu"),

tf.keras.layers.Dense(26, activation="softmax") # 26 letters A-Z

])

model.compile(

loss="sparse\_categorical\_crossentropy",

optimizer=Adam(learning\_rate=0.001),

metrics=["accuracy"]

)

model.summary()

return model

def plot\_history(train, val, title):

epochs = range(len(train))

plt.figure()

plt.plot(epochs, train, label="train")

plt.plot(epochs, val, label="val")

plt.title(title)

plt.legend(loc="best")

plt.show()

plt.pause(1.0)

def classify\_images(fn\_arr, model):

for fn in fn\_arr:

path = "datasets/" + fn

img = keras\_image.load\_img(path, target\_size=(28, 28), color\_mode="grayscale")

x = keras\_image.img\_to\_array(img)

x = np.expand\_dims(x, axis=0)

num\_to\_alphabet = np.array([

'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',

'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R',

'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'

])

image\_i = np.vstack([x])

classes = model.predict(image\_i, batch\_size=10)

print(classes[0])

class\_label = num\_to\_alphabet[np.argmax(classes[0])]

print(fn + " is a letter {:s}".format(class\_label))

if \_\_name\_\_ == "\_\_main\_\_":

# Load the dataset

training\_images, training\_labels = get\_data(os.getcwd() + "/datasets/sign\_mnist\_train.csv")

validation\_images, validation\_labels = get\_data(os.getcwd() + "/datasets/sign\_mnist\_test.csv")

print(training\_images.shape)

print(training\_labels.shape)

print(validation\_images.shape)

print(validation\_labels.shape)

training\_images = np.expand\_dims(training\_images, axis=-1)

validation\_images = np.expand\_dims(validation\_images, axis=-1)

print(training\_images.shape)

print(validation\_images.shape)

print(np.max(training\_labels), np.min(training\_labels))

print(np.max(validation\_labels), np.min(validation\_labels))

# Plot one of the images and its label

image\_num = 2

plot\_one\_image(training\_images[image\_num, :, :, 0], training\_labels[image\_num])

# Data pre-processing with ImageDataGenerator

training\_generator, validation\_generator = do\_data\_preprocessing(

training\_images, training\_labels,

validation\_images, validation\_labels

)

# Build a CNN model

cnn\_model = create\_cnn\_model()

# Train the model

history = cnn\_model.fit(

training\_generator,

steps\_per\_epoch=len(training\_images) // 20,

epochs=50,

validation\_data=validation\_generator,

validation\_steps=len(validation\_images) // 20

)

# Evaluate the model

cnn\_model.evaluate(validation\_generator)

# Plot training history

acc = history.history["accuracy"]

val\_acc = history.history["val\_accuracy"]

loss = history.history["loss"]

val\_loss = history.history["val\_loss"]

plot\_history(acc, val\_acc, "Training and validation accuracy")

plot\_history(loss, val\_loss, "Training and validation loss")

# Test on sample images

fn\_arr = [

"alphabet-letter-C-1298289\_640.png",

"alphabet-letter-D-1298315\_640.png",

"alphabet-letter-Y-1298311\_640.png",

"sign-language-letter-A-28717\_640.png"

]

classify\_images(fn\_arr, cnn\_model)

1. Hasil Percobaan
2. Listing 2.3. fashion\_classifier\_with\_callback.py

A screen shot of a computer

AI-generated content may be incorrect.

1. Listing 2.4. fashion\_classifier\_with\_cnn.py

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

1. Listing 2.5. face\_expression\_classifier\_with\_cnn.py

A screenshot of a video game

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

1. Listing 3.1. cats\_and\_dogs\_classifier\_with\_cnn.py

A screenshot of a computer

AI-generated content may be incorrect.

A black screen with white text

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A graph on a computer screen

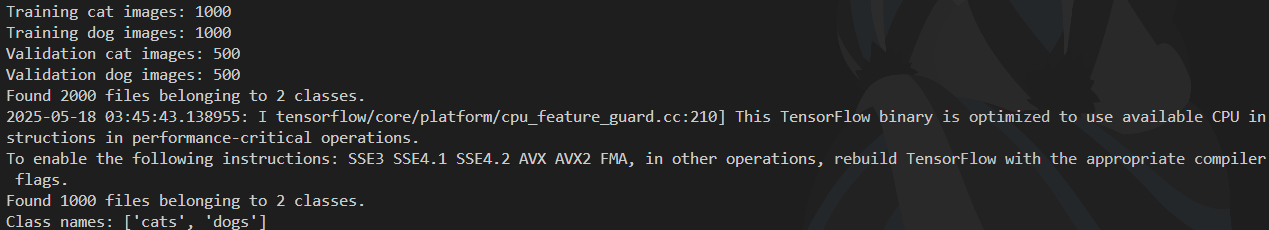
AI-generated content may be incorrect. A graph on a computer screen

AI-generated content may be incorrect.

A computer screen shot of a program

AI-generated content may be incorrect.

1. Listing 3.2. cats\_and\_dogs\_classifier\_with\_imagedatagen\_and\_dropout.py



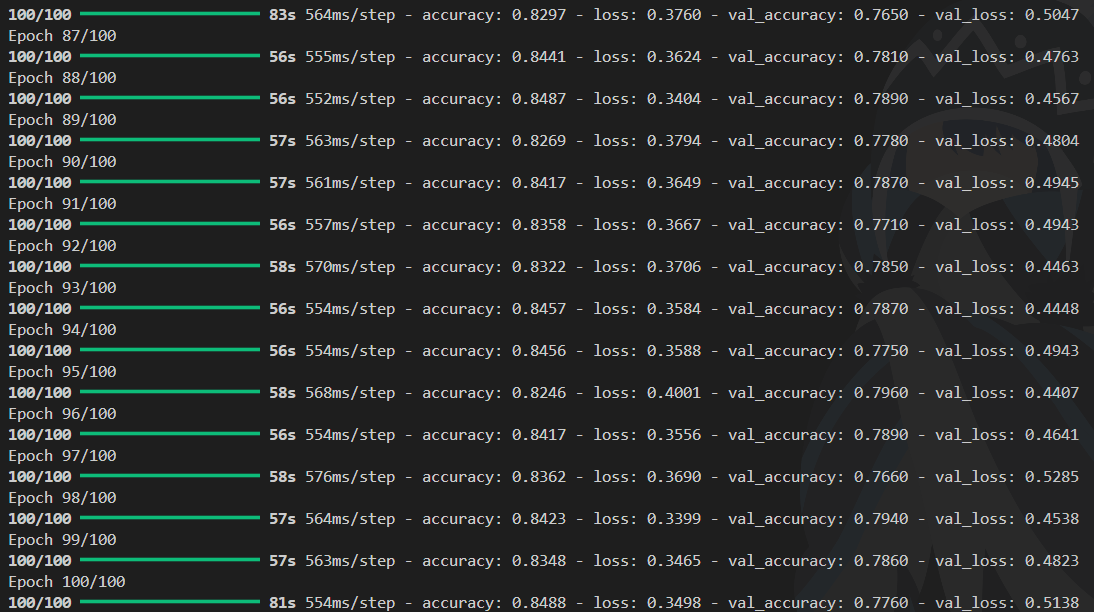
A screenshot of a computer program

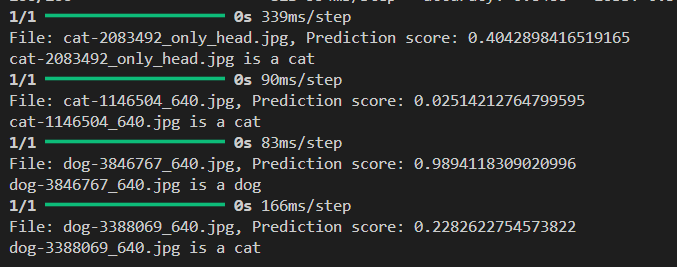
AI-generated content may be incorrect.

A graph on a computer screen

AI-generated content may be incorrect. A screenshot of a graph

AI-generated content may be incorrect.





1. Listing 3.3. cats\_and\_dogs\_classifier\_with\_transfer\_learning.py

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer screen

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

1. Listing 3.4. hand\_sign\_language\_classifier.py

A screen shot of a hand

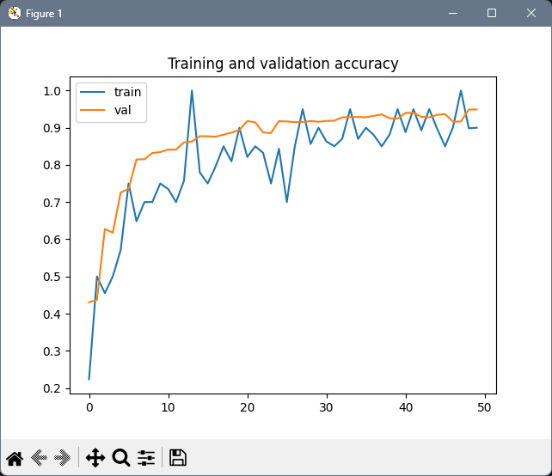
AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.\

A screenshot of a computer program

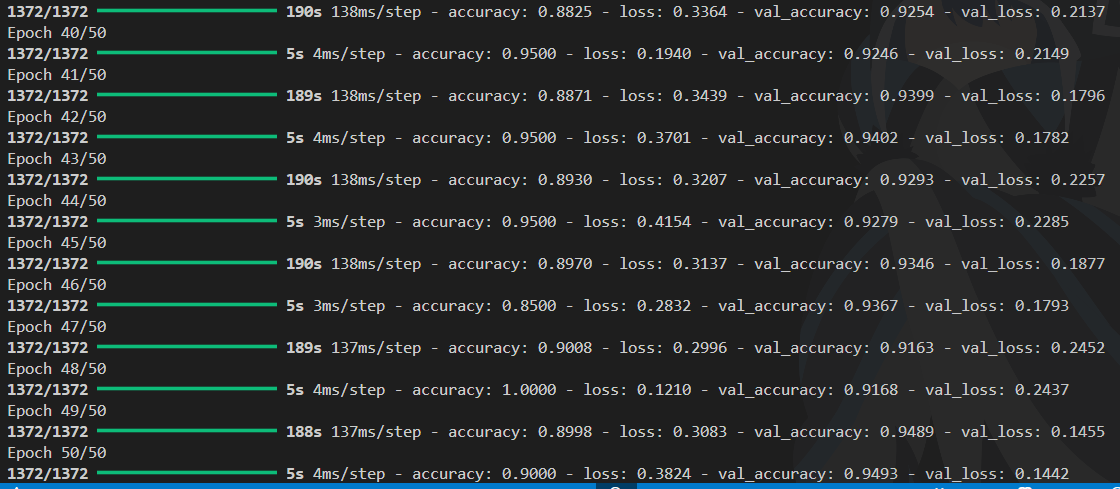
AI-generated content may be incorrect.

 A graph with blue and orange lines

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.



1. Analisa

Praktikum ini mencakup beberapa program klasifikasi menggunakan TensorFlow dengan berbagai dataset dan model yang berbeda. Program pertama adalah pelatihan model jaringan saraf tiruan sederhana pada dataset Fashion MNIST. Program ini menggunakan callback khusus untuk menghentikan pelatihan secara otomatis ketika akurasi sudah melebihi 60%, sehingga proses menjadi lebih efisien. Data gambar berukuran 28x28 piksel dinormalisasi agar piksel berada di rentang 0 hingga 1. Model dibangun dengan lapisan Flatten untuk mengubah input 2D menjadi 1D, diikuti Dense layer dengan 512 neuron dan aktivasi ReLU, serta output layer dengan 10 neuron menggunakan aktivasi softmax untuk klasifikasi multi-kelas. Model dilatih dengan optimizer Adam dan fungsi loss sparse\_categorical\_crossentropy selama maksimal 10 epoch, namun pelatihan bisa berhenti lebih awal sesuai callback.

Program kedua menggunakan arsitektur Convolutional Neural Network (CNN) untuk klasifikasi gambar Fashion MNIST. Setelah data dinormalisasi dan ditambahkan dimensi channel agar sesuai input CNN, model dibangun dengan dua lapisan Conv2D berfilter 64 ukuran 3x3, diikuti MaxPooling2D untuk mengurangi dimensi fitur. Selanjutnya data di-flatten dan dilanjutkan ke dense layer untuk klasifikasi. Program ini juga menyediakan fungsi visualisasi untuk memperlihatkan proses convolution dan max pooling pada lapisan awal, sehingga pengguna dapat memahami bagaimana fitur gambar diekstraksi oleh model.

Program ketiga fokus pada klasifikasi ekspresi wajah “happy” dan “sad” menggunakan CNN. Dataset wajah berlabel diekstrak dari file ZIP dan diproses dengan ImageDataGenerator untuk normalisasi serta pembagian data pelatihan dan validasi. Model CNN terdiri dari beberapa lapisan konvolusi dan pooling, lalu flatten dan dua dense layer, dengan output sigmoid untuk klasifikasi biner. Callback dipasang agar pelatihan berhenti ketika akurasi mencapai 99%. Setelah pelatihan selesai, model digunakan untuk memprediksi ekspresi pada gambar uji, dengan hasil prediksi dikategorikan berdasarkan threshold 0.5. Program juga menampilkan gambar sampel dari masing-masing kelas sebelum pelatihan.

Program keempat membangun model CNN untuk membedakan antara gambar kucing dan anjing. Dataset diekstrak dan dipisahkan menjadi data pelatihan dan validasi, lalu diproses menggunakan ImageDataGenerator untuk penskalaan piksel. Model CNN terdiri dari tiga lapisan konvolusi dan pooling, kemudian flatten dan dua dense layer, dengan aktivasi sigmoid pada output untuk klasifikasi biner. Selain melatih model, program ini juga memvisualisasikan grafik akurasi dan loss selama proses pelatihan serta menampilkan representasi feature map dari setiap lapisan CNN untuk memperlihatkan proses ekstraksi fitur.

Program kelima merupakan pengembangan dari program sebelumnya dengan penambahan augmentasi data seperti rotasi, zoom, dan flip horizontal untuk meningkatkan kemampuan model mengenali data baru dan mencegah overfitting. Gambar diresize menjadi 150x150 piksel dan diproses dalam batch berukuran 20. Model CNN yang digunakan juga memiliki lapisan dropout untuk mengurangi overfitting. Setelah pelatihan, akurasi dan loss divisualisasikan dan terdapat fungsi untuk menguji prediksi model pada gambar individual di luar dataset pelatihan.

Program keenam mengaplikasikan transfer learning dengan menggunakan model VGG16 tanpa top layer untuk klasifikasi kucing dan anjing. Dataset diproses menggunakan ImageDataGenerator dengan augmentasi opsional. Bobot model VGG16 yang sudah dilatih sebelumnya dimuat dan lapisan-lapisan awal dibekukan agar tidak ikut dilatih ulang. Di atas model tersebut ditambahkan lapisan dropout, flatten, dan dua dense layer dengan output sigmoid untuk klasifikasi biner. Model ini dilatih selama 20 epoch dan hasil pelatihan divisualisasikan dalam grafik. Tersedia juga fungsi untuk menguji model pada gambar baru dari folder tertentu.

Terakhir, program ketujuh mengimplementasikan klasifikasi huruf alfabet dari dataset Sign Language MNIST. Data diambil dari file CSV dan diolah menjadi gambar 28x28 piksel grayscale. Augmentasi data dilakukan untuk mengurangi risiko overfitting. Model CNN yang dibangun terdiri dari lapisan Conv2D, MaxPooling, Dropout, Flatten, dan dense layer dengan 26 neuron output dan fungsi aktivasi softmax untuk klasifikasi multi-kelas. Model dilatih selama 50 epoch dengan validasi, dan program menyediakan fungsi untuk mengklasifikasikan gambar eksternal serta menampilkan hasil prediksi huruf yang paling mungkin.

1. Kesimpulan

Kesimpulannya, praktikum ini memperlihatkan berbagai pendekatan dalam pembangunan model klasifikasi gambar menggunakan TensorFlow, mulai dari jaringan saraf tiruan sederhana hingga Convolutional Neural Network (CNN) yang lebih kompleks. Penggunaan callback untuk menghentikan pelatihan secara otomatis dan teknik augmentasi data terbukti efektif dalam meningkatkan efisiensi pelatihan serta kemampuan generalisasi model. Transfer learning dengan model VGG16 juga menunjukkan bagaimana pemanfaatan model pra-latih dapat mempercepat dan memperbaiki hasil klasifikasi. Seluruh program menekankan pentingnya preprocessing data, seperti normalisasi dan augmentasi, serta penggunaan arsitektur yang sesuai dengan jenis data dan tugas klasifikasi yang dihadapi.