Praktikum Sistem Cerdas

NRP	: 3223600019
Nama	: Muhammad Bimo Fachrizky
Materi	: Membuat Aplikasi Deep Learning tensorflow 2
Tanggal	: Senin, 05 Mei 2025

Praktikum 10

Membuat Aplikasi Deep Learning tensorflow 2

I. 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

II. Langkah percobaan

1. 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])
```

2. 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)
```

3. 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)
```

4. 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")
```

5. 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}")
```

6. 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)
```

7. 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)
```

III. Hasil Percobaan

1. Listing 2.3. fashion_classifier_with_callback.py

```
2025-05-17 22:24:21.045840: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized t
o use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with th
e appropriate compiler flags.
Epoch 1/10
1871/1875 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7934 - loss: 0.5841
Reached 60% accuracy so cancelling training!
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 17s 9ms/step - accuracy: 0.7935 - loss: 0.5838
```

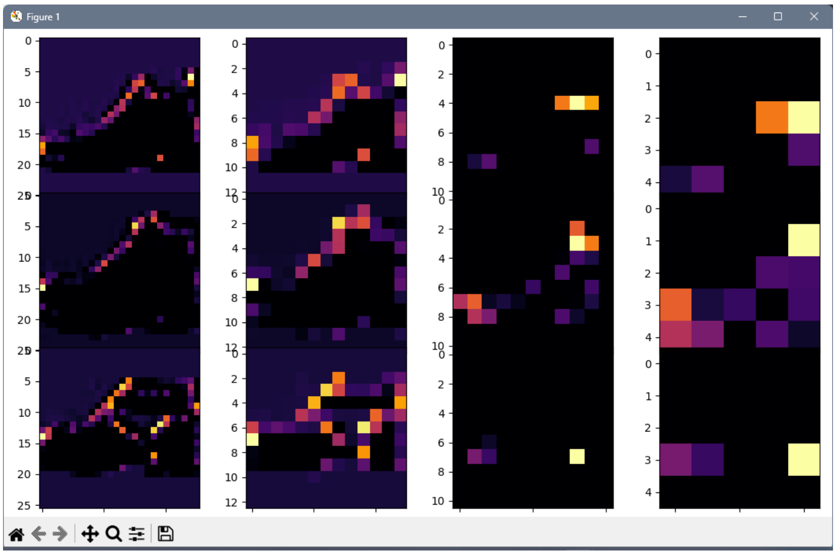
2. Listing 2.4. fashion_classifier_with_cnn.py

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 64)	640
max_pooling2d (MaxPooling2D)	(None, 13, 13, 64)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	36,928
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dense (Dense)	(None, 128)	204,928
dense_1 (Dense)	(None, 10)	1,290

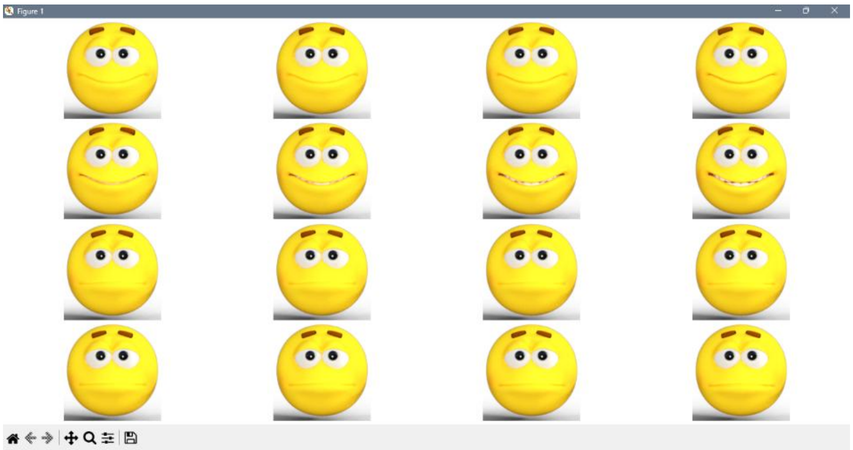
```
Total params: 243,786 (952.29 KB)
Trainable params: 243,786 (952.29 KB)
Non-trainable params: 0 (0.00 B)
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 64s 32ms/step - accuracy: 0.7836 - loss: 0.6020
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 52s 28ms/step - accuracy: 0.8880 - loss: 0.3047
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 52s 28ms/step - accuracy: 0.9082 - loss: 0.2521
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 51s 27ms/step - accuracy: 0.9186 - loss: 0.2219
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 50s 27ms/step - accuracy: 0.9310 - loss: 0.1870
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 50s 27ms/step - accuracy: 0.9375 - loss: 0.1672
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 52s 28ms/step - accuracy: 0.9458 - loss: 0.1417
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 84s 29ms/step - accuracy: 0.9523 - loss: 0.1262
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 55s 29ms/step - accuracy: 0.9588 - loss: 0.1104
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 52s 28ms/step - accuracy: 0.9627 - loss: 0.0968
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - accuracy: 0.9096 - loss: 0.3253
```



```
[[9 2 1 1 6 1 4 6 5 7]
 [4 5 7 3 4 1 2 4 8 0]
 [2 5 7 9 1 4 6 0 9 3]
 [8 8 3 3 8 0 7 5 7 9]
 [6 1 3 7 6 7 2 1 2 2]
 [4 4 5 8 2 2 8 4 8 0]
 [7 7 8 5 1 1 2 3 9 8]
 [7 0 2 6 2 3 1 2 8 4]
 [1 8 5 9 5 0 3 2 0 6]
 [5 3 6 7 1 8 0 1 4 2]]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 189ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 89ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 78ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step
```

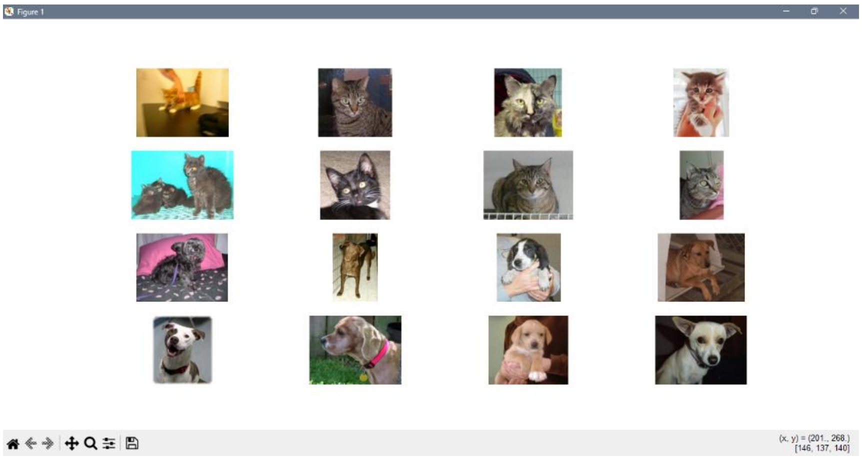
3. Listing 2.5. face_expression_classifier_with_cnn.py



```
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 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_2 (Conv2D)               │ (None, 34, 34, 64)     │        18,496 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_2 (MaxPooling2D)  │ (None, 17, 17, 64)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten (Flatten)               │ (None, 18496)          │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                   │ (None, 512)            │     9,470,464 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 1)              │           513 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 9,494,561 (36.22 MB)
 Trainable params: 9,494,561 (36.22 MB)
 Non-trainable params: 0 (0.00 B)
```

```
Epoch 1/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 8s 370ms/step - accuracy: 0.7171 - loss: 0.6252 - val_accuracy: 0.7500 - val_loss: 0.5721
Epoch 2/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 141ms/step - accuracy: 1.0000 - loss: 0.1850
Reached 99.00% accuracy so cancelling training!
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 164ms/step - accuracy: 1.0000 - loss: 0.1751 - val_accuracy: 0.6875 - val_loss: 1.2106
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 204ms/step
Prediction score for beauty-1132617_640.jpg: 0.9826841354370117
beauty-1132617_640.jpg is happy
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step
Prediction score for girl-2961959_640.jpg: 0.9709166884422302
girl-2961959_640.jpg is happy
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step
Prediction score for woman-2126727_640.jpg: 0.9999503493309021
woman-2126727_640.jpg is happy
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
Prediction score for beautiful-18279_640.jpg: 0.9998217225074768
beautiful-18279_640.jpg is happy
```

4. Listing 3.1. cats_and_dogs_classifier_with_cnn.py




```
['cat.0.jpg', 'cat.1.jpg', 'cat.10.jpg', 'cat.100.jpg', 'cat.101.jpg', 'cat.102.jpg', 'cat.103.jpg', 'cat.104.jpg', 'cat.105.jpg', 'cat.106.jpg']
['dog.0.jpg', 'dog.1.jpg', 'dog.10.jpg', 'dog.100.jpg', 'dog.101.jpg', 'dog.102.jpg', 'dog.103.jpg', 'dog.104.jpg', 'dog.105.jpg', 'dog.106.jpg']
total training cat images : 1000
total training dog images : 1000
total validation cat images: 500
total validation dog images: 500
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
2025-05-18 03:23:49.314680: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 150, 150, 3)	0
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
conv2d_2 (Conv2D)	(None, 34, 34, 64)	18,496
max_pooling2d_2 (MaxPooling2D)	(None, 17, 17, 64)	0
flatten (Flatten)	(None, 18496)	0
dense (Dense)	(None, 512)	9,470,464
dense_1 (Dense)	(None, 1)	513

Total params: 9,494,561 (36.22 MB)
Trainable params: 9,494,561 (36.22 MB)
Non-trainable params: 0 (0.00 B)

```
Epoch 3/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 36s 363ms/step - accuracy: 0.6561 - loss: 0.6344 - val_accuracy: 0.6400 - val_loss: 0.6656
Epoch 4/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 27s 273ms/step - accuracy: 0.7185 - loss: 0.5672 - val_accuracy: 0.6370 - val_loss: 0.6450
Epoch 5/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 28s 280ms/step - accuracy: 0.7925 - loss: 0.4470 - val_accuracy: 0.6680 - val_loss: 0.6672
Epoch 6/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 29s 286ms/step - accuracy: 0.8562 - loss: 0.3319 - val_accuracy: 0.6330 - val_loss: 0.7525
Epoch 7/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 29s 289ms/step - accuracy: 0.9213 - loss: 0.1949 - val_accuracy: 0.6500 - val_loss: 1.0080
Epoch 8/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 29s 291ms/step - accuracy: 0.9561 - loss: 0.1233 - val_accuracy: 0.6640 - val_loss: 1.1302
Epoch 9/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 28s 276ms/step - accuracy: 0.9879 - loss: 0.0464 - val_accuracy: 0.6660 - val_loss: 1.6276
Epoch 10/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 27s 268ms/step - accuracy: 0.9946 - loss: 0.0168 - val_accuracy: 0.6610 - val_loss: 1.6432
Epoch 11/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 27s 273ms/step - accuracy: 0.9989 - loss: 0.0089 - val_accuracy: 0.6710 - val_loss: 1.8931
Epoch 12/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 27s 273ms/step - accuracy: 0.9843 - loss: 0.0423 - val_accuracy: 0.6580 - val_loss: 1.6422
Epoch 13/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 28s 275ms/step - accuracy: 0.9943 - loss: 0.0197 - val_accuracy: 0.6480 - val_loss: 1.6995
Epoch 14/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 27s 273ms/step - accuracy: 0.9922 - loss: 0.0274 - val_accuracy: 0.6550 - val_loss: 1.9293
Epoch 15/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 29s 285ms/step - accuracy: 0.9989 - loss: 0.0030 - val_accuracy: 0.6520 - val_loss: 2.0560
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 282ms/step
```

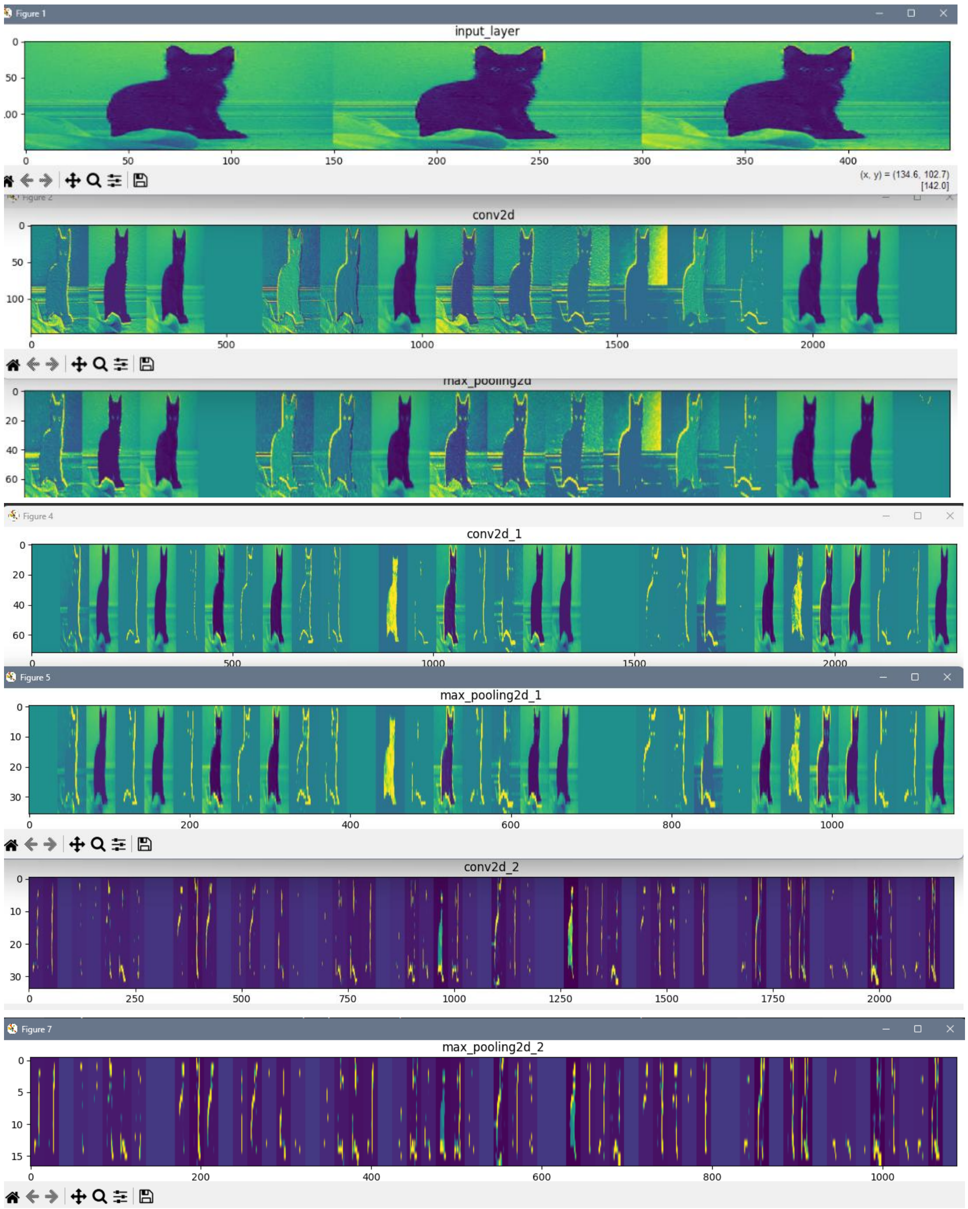


Figure 1
input_layer
(x, y) = (134.6, 102.7)
[142.0]
conv2d
max_pooling2d
Figure 4
conv2d_1
Figure 5
max_pooling2d_1
conv2d_2
Figure 7
max_pooling2d_2

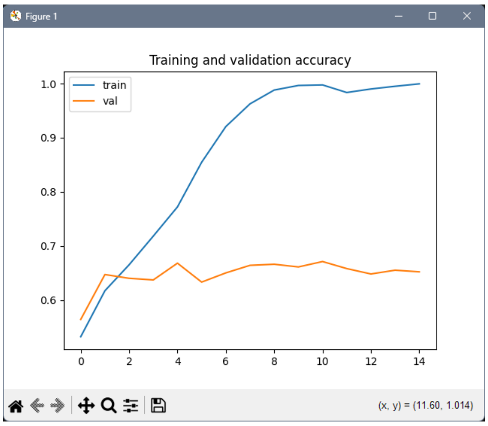


Figure 1
Training and validation accuracy
train
val
(x, y) = (11.60, 1.014)

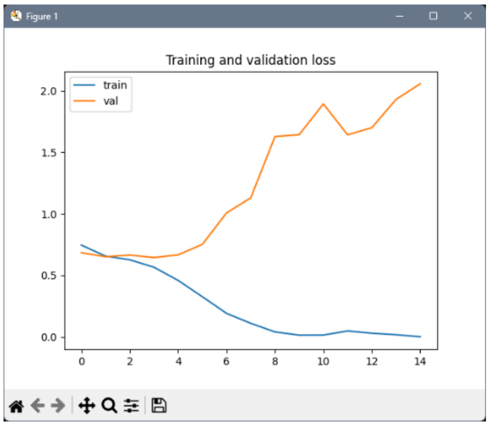


Figure 1
Training and validation loss
train
val

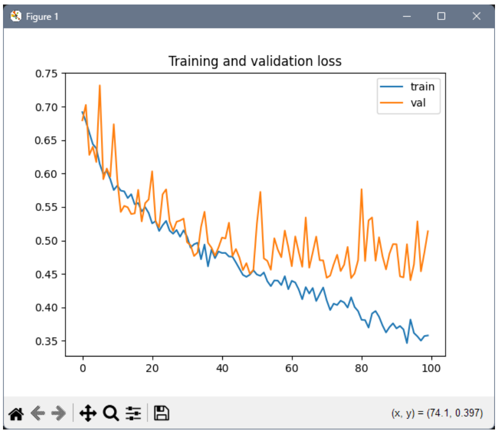
```
[0.]
cat-2083492_only_head.jpg is a cat
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 105ms/step
[1.]
cat-1146504_640.jpg is a dog
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 108ms/step
[1.]
dog-3846767_640.jpg is a dog
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 102ms/step
[1.]
dog-3388069_640.jpg is a dog
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 429ms/step
```

5. Listing 3.2. cats_and_dogs_classifier_with_imagedatagen_and_dropout.py

```
Training cat images: 1000
Training dog images: 1000
Validation cat images: 500
Validation dog images: 500
Found 2000 files belonging to 2 classes.
2025-05-18 03:45:43.138955: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU in
structions in performance-critical operations.
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler
 flags.
Found 1000 files belonging to 2 classes.
Class names: ['cats', 'dogs']
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_2 (Conv2D)	(None, 34, 34, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 17, 17, 128)	0
conv2d_3 (Conv2D)	(None, 15, 15, 128)	147,584
max_pooling2d_3 (MaxPooling2D)	(None, 7, 7, 128)	0
dropout (Dropout)	(None, 7, 7, 128)	0
flatten (Flatten)	(None, 6272)	0
dense (Dense)	(None, 512)	3,211,776
dense_1 (Dense)	(None, 1)	513

```
Total params: 3,453,121 (13.17 MB)
Trainable params: 3,453,121 (13.17 MB)
```



```
100/100 ━━━━━━━━━━━━━━━━━━━━ 83s 564ms/step - accuracy: 0.8297 - loss: 0.3760 - val_accuracy: 0.7650 - val_loss: 0.5047
Epoch 87/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 56s 555ms/step - accuracy: 0.8441 - loss: 0.3624 - val_accuracy: 0.7810 - val_loss: 0.4763
Epoch 88/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 56s 552ms/step - accuracy: 0.8487 - loss: 0.3404 - val_accuracy: 0.7890 - val_loss: 0.4567
Epoch 89/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 57s 563ms/step - accuracy: 0.8269 - loss: 0.3794 - val_accuracy: 0.7780 - val_loss: 0.4804
Epoch 90/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 57s 561ms/step - accuracy: 0.8417 - loss: 0.3649 - val_accuracy: 0.7870 - val_loss: 0.4945
Epoch 91/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 56s 557ms/step - accuracy: 0.8358 - loss: 0.3667 - val_accuracy: 0.7710 - val_loss: 0.4943
Epoch 92/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 58s 570ms/step - accuracy: 0.8322 - loss: 0.3706 - val_accuracy: 0.7850 - val_loss: 0.4463
Epoch 93/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 56s 554ms/step - accuracy: 0.8457 - loss: 0.3584 - val_accuracy: 0.7870 - val_loss: 0.4448
Epoch 94/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 56s 554ms/step - accuracy: 0.8456 - loss: 0.3588 - val_accuracy: 0.7750 - val_loss: 0.4943
Epoch 95/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 58s 568ms/step - accuracy: 0.8246 - loss: 0.4001 - val_accuracy: 0.7960 - val_loss: 0.4407
Epoch 96/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 56s 554ms/step - accuracy: 0.8417 - loss: 0.3556 - val_accuracy: 0.7890 - val_loss: 0.4641
Epoch 97/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 58s 576ms/step - accuracy: 0.8362 - loss: 0.3690 - val_accuracy: 0.7660 - val_loss: 0.5285
Epoch 98/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 57s 564ms/step - accuracy: 0.8423 - loss: 0.3399 - val_accuracy: 0.7940 - val_loss: 0.4538
Epoch 99/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 57s 563ms/step - accuracy: 0.8348 - loss: 0.3465 - val_accuracy: 0.7860 - val_loss: 0.4823
Epoch 100/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 81s 554ms/step - accuracy: 0.8488 - loss: 0.3498 - val_accuracy: 0.7760 - val_loss: 0.5138
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 339ms/step
File: cat-2083492_only_head.jpg, Prediction score: 0.4042898416519165
cat-2083492_only_head.jpg is a cat
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 90ms/step
File: cat-1146504_640.jpg, Prediction score: 0.02514212764799595
cat-1146504_640.jpg is a cat
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step
File: dog-3846767_640.jpg, Prediction score: 0.9894118309020996
dog-3846767_640.jpg is a dog
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 166ms/step
File: dog-3388069_640.jpg, Prediction score: 0.2282622754573822
dog-3388069_640.jpg is a cat
```

6. Listing 3.3. cats_and_dogs_classifier_with_transfer_learning.py

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 150, 150, 3)	0
block1_conv1 (Conv2D)	(None, 150, 150, 64)	1,792
block1_conv2 (Conv2D)	(None, 150, 150, 64)	36,928
block1_pool (MaxPooling2D)	(None, 75, 75, 64)	0
block2_conv1 (Conv2D)	(None, 75, 75, 128)	73,856
block2_conv2 (Conv2D)	(None, 75, 75, 128)	147,584
block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Conv2D)	(None, 37, 37, 256)	295,168
block3_conv2 (Conv2D)	(None, 37, 37, 256)	590,080
block3_conv3 (Conv2D)	(None, 37, 37, 256)	590,080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Conv2D)	(None, 18, 18, 512)	1,180,160

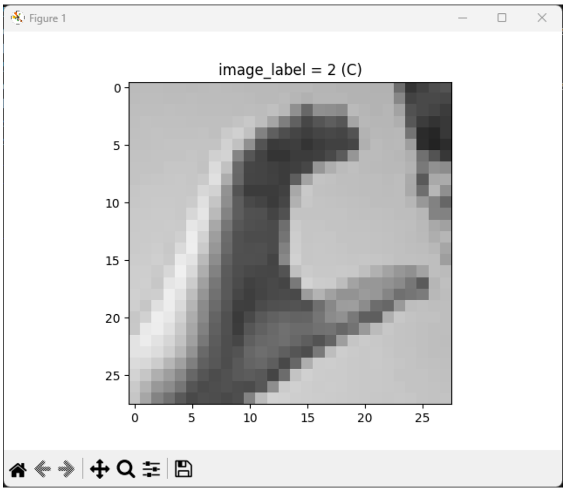
```
block4_conv2 (Conv2D)          (None, 18, 18, 512)       2,359,808
block4_conv3 (Conv2D)          (None, 18, 18, 512)       2,359,808
block4_pool (MaxPooling2D)     (None, 9, 9, 512)                 0
block5_conv1 (Conv2D)          (None, 9, 9, 512)         2,359,808
block5_conv2 (Conv2D)          (None, 9, 9, 512)         2,359,808
block5_conv3 (Conv2D)          (None, 9, 9, 512)         2,359,808
block5_pool (MaxPooling2D)     (None, 4, 4, 512)                 0

 Total params: 14,714,688 (56.13 MB)
 Trainable params: 0 (0.00 B)
 Non-trainable params: 14,714,688 (56.13 MB)
last layer output shape: (None, 4, 4, 512)
```

```
Epoch 10/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 430s 4s/step - accuracy: 0.8584 - loss: 0.3132 - val_accuracy: 0.8910 - val_loss: 0.2545
Epoch 11/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 430s 4s/step - accuracy: 0.8690 - loss: 0.2983 - val_accuracy: 0.8970 - val_loss: 0.2615
Epoch 12/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 429s 4s/step - accuracy: 0.8807 - loss: 0.2891 - val_accuracy: 0.8800 - val_loss: 0.3108
Epoch 13/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 429s 4s/step - accuracy: 0.8447 - loss: 0.3436 - val_accuracy: 0.8920 - val_loss: 0.2714
Epoch 14/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 429s 4s/step - accuracy: 0.8592 - loss: 0.3062 - val_accuracy: 0.8830 - val_loss: 0.2721
Epoch 15/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 432s 4s/step - accuracy: 0.8663 - loss: 0.3312 - val_accuracy: 0.8910 - val_loss: 0.2570
Epoch 16/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 431s 4s/step - accuracy: 0.8675 - loss: 0.2949 - val_accuracy: 0.8870 - val_loss: 0.2898
Epoch 17/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 433s 4s/step - accuracy: 0.8730 - loss: 0.2980 - val_accuracy: 0.8930 - val_loss: 0.2571
Epoch 18/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 438s 4s/step - accuracy: 0.8495 - loss: 0.3027 - val_accuracy: 0.8870 - val_loss: 0.2623
Epoch 19/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 440s 4s/step - accuracy: 0.8640 - loss: 0.3062 - val_accuracy: 0.8910 - val_loss: 0.2582
Epoch 20/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 430s 4s/step - accuracy: 0.8578 - loss: 0.3418 - val_accuracy: 0.8860 - val_loss: 0.2458
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 706ms/step
```

```
cat-2083492_only_head.jpg prediction score: 9.425148044552749e-38
cat-2083492_only_head.jpg is a cat
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 396ms/step
cat-1146504_640.jpg prediction score: 0.0
cat-1146504_640.jpg is a cat
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 401ms/step
dog-3846767_640.jpg prediction score: 1.0
dog-3846767_640.jpg is a dog
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 402ms/step
dog-3388069_640.jpg prediction score: 1.0
dog-3388069_640.jpg is a dog
```

7. Listing 3.4. hand_sign_language_classifier.py

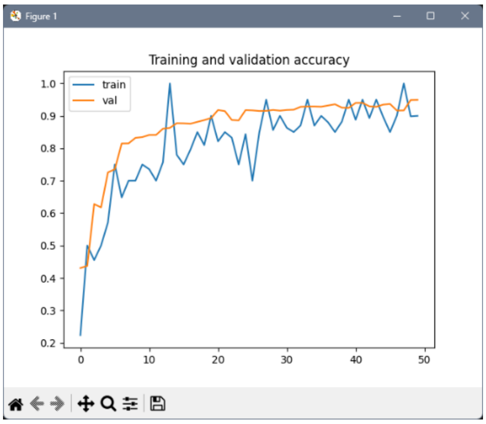


```
processing: 100.00 %
processing: 100.00 %
(27455, 28, 28)
(27455,)
(7172, 28, 28)
(7172,)
(27455, 28, 28, 1)
(7172, 28, 28, 1)
24.0 0.0
24.0 0.0
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 64)	640
max_pooling2d (MaxPooling2D)	(None, 13, 13, 64)	0
dropout (Dropout)	(None, 13, 13, 64)	0
flatten (Flatten)	(None, 10816)	0
dense (Dense)	(None, 1024)	11,076,608
dense_1 (Dense)	(None, 26)	26,650

Total params: 11,103,898 (42.36 MB)
Trainable params: 11,103,898 (42.36 MB)
Non-trainable params: 0 (0.00 B)




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step
[0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
alphabet-letter-C-1298289_640.png is a letter I
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
 0. 0.]
alphabet-letter-D-1298315_640.png is a letter R
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 68ms/step
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
 0. 0.]
alphabet-letter-Y-1298311_640.png is a letter S
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
sign-language-letter-A-28717_640.png is a letter N
```

```
1372/1372 ━━━━━━━━━━━━━━━━━━━━ 190s 138ms/step - accuracy: 0.8825 - loss: 0.3364 - val_accuracy: 0.9254 - val_loss: 0.2137
Epoch 40/50
1372/1372 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9500 - loss: 0.1940 - val_accuracy: 0.9246 - val_loss: 0.2149
Epoch 41/50
1372/1372 ━━━━━━━━━━━━━━━━━━━━ 189s 138ms/step - accuracy: 0.8871 - loss: 0.3439 - val_accuracy: 0.9399 - val_loss: 0.1796
Epoch 42/50
1372/1372 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9500 - loss: 0.3701 - val_accuracy: 0.9402 - val_loss: 0.1782
Epoch 43/50
1372/1372 ━━━━━━━━━━━━━━━━━━━━ 190s 138ms/step - accuracy: 0.8930 - loss: 0.3207 - val_accuracy: 0.9293 - val_loss: 0.2257
Epoch 44/50
1372/1372 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9500 - loss: 0.4154 - val_accuracy: 0.9279 - val_loss: 0.2285
Epoch 45/50
1372/1372 ━━━━━━━━━━━━━━━━━━━━ 190s 138ms/step - accuracy: 0.8970 - loss: 0.3137 - val_accuracy: 0.9346 - val_loss: 0.1877
Epoch 46/50
1372/1372 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8500 - loss: 0.2832 - val_accuracy: 0.9367 - val_loss: 0.1793
Epoch 47/50
1372/1372 ━━━━━━━━━━━━━━━━━━━━ 189s 137ms/step - accuracy: 0.9008 - loss: 0.2996 - val_accuracy: 0.9163 - val_loss: 0.2452
Epoch 48/50
1372/1372 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 1.0000 - loss: 0.1210 - val_accuracy: 0.9168 - val_loss: 0.2437
Epoch 49/50
1372/1372 ━━━━━━━━━━━━━━━━━━━━ 188s 137ms/step - accuracy: 0.8998 - loss: 0.3083 - val_accuracy: 0.9489 - val_loss: 0.1455
Epoch 50/50
1372/1372 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9000 - loss: 0.3824 - val_accuracy: 0.9493 - val_loss: 0.1442
```

IV. 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.

V. 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.