Import Library

In [2]:
```
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
import numpy as np
import os

tf.random.set_seed(42)
print("Tensorflow version:", tf.__version__)
```

```
2025-10-27 01:49:10.413086: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-10-27 01:49:10.433782: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT when one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
E0000 00:00:1761504550.456768  424008 cuda_dnn.cc:8310] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been registered
E0000 00:00:1761504550.463949  424008 cuda_blas.cc:1418] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been registered
2025-10-27 01:49:10.488327: 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: AVX2 AVX512F AVX512_VNNI AVX512_BF16 AVX512_FP16 AVX_VNNI AMX_TILE AMX_INT8 AMX_BF16 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Tensorflow version: 2.18.0
```

Load Dataset Utama

In [3]:
```
train_dir = '../dataset_split/train'
val_dir = '../dataset_split/validation'
test_dir = '../dataset_split/test'

train_ds = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(128, 128),
    batch_size=16
)

val_ds = tf.keras.utils.image_dataset_from_directory(
    val_dir,
    image_size=(128, 128),
    batch_size=16
)

test_ds = tf.keras.utils.image_dataset_from_directory(
    test_dir,
    image_size=(128, 128),
```

```
    batch_size=16
)
```

```
Found 240 files belonging to 4 classes.
Using 192 files for training.
I0000 00:00:1761504555.051340  424008 gpu_device.cc:2022] Created device /job:loc
alhost/replica:0/task:0/device:GPU:0 with 6748 MB memory:  -> device: 0, name: NV
IDIA L40S, pci bus id: 0000:16:00.0, compute capability: 8.9
I0000 00:00:1761504555.053099  424008 gpu_device.cc:2022] Created device /job:loc
alhost/replica:0/task:0/device:GPU:1 with 19965 MB memory:  -> device: 1, name: N
VIDIA L40S, pci bus id: 0000:be:00.0, compute capability: 8.9
Found 32 files belonging to 4 classes.
Found 28 files belonging to 4 classes.
```

Normalisasi dan Optimasi Pipeline

In [4]:
```
normalization_layer = layers.Rescaling(1./255)

train_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
val_ds = val_ds.map(lambda x, y: (normalization_layer(x), y))
test_ds = test_ds.map(lambda x, y: (normalization_layer(x), y))
```

Mendefinisikan CNN

In [6]:
```
model = models.Sequential([
    layers.Conv2D(24, (3, 3), activation='relu', input_shape=(128, 128, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(36, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(4, activation='softmax')
])

model.summary()
```

```
/opt/tljh/user/envs/dltf/lib/python3.12/site-packages/keras/src/layers/convolutio
nal/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argum
ent to a layer. When using Sequential models, prefer using an `Input(shape)` obje
ct as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 126, 24)	672
max_pooling2d (MaxPooling2D)	(None, 63, 63, 24)	0
conv2d_1 (Conv2D)	(None, 61, 61, 36)	7,812
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 36)	0
flatten (Flatten)	(None, 32400)	0
dense (Dense)	(None, 128)	4,147,328
dense_1 (Dense)	(None, 4)	516

Total params: 4,156,328 (15.86 MB)

Trainable params: 4,156,328 (15.86 MB)

Non-trainable params: 0 (0.00 B)

Kompilasi Model

In [7]:
```
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

Training Model

In [8]:
```
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=15
)
```

```
Epoch 1/15
WARNING: All log messages before absl::InitializeLog() is called are written to S
TDERR
I0000 00:00:1761504606.402516  424453 service.cc:148] XLA service 0x7f1fd400a150
initialized for platform CUDA (this does not guarantee that XLA will be used). De
vices:
I0000 00:00:1761504606.402547  424453 service.cc:156]   StreamExecutor device
(0): NVIDIA L40S, Compute Capability 8.9
I0000 00:00:1761504606.402550  424453 service.cc:156]   StreamExecutor device
(1): NVIDIA L40S, Compute Capability 8.9
2025-10-27 01:50:06.448954: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir
_util.cc:268] disabling MLIR crash reproducer, set env var `MLIR_CRASH_REPRODUCER
_DIRECTORY` to enable.
I0000 00:00:1761504606.570652  424453 cuda_dnn.cc:529] Loaded cuDNN version 91002
 1/12 ━━━━━━━━━━━━━━━━━━━━ 26s 2s/step - accuracy: 0.3125 - loss: 1.3962
I0000 00:00:1761504607.927696  424453 device_compiler.h:188] Compiled cluster usi
ng XLA!  This line is logged at most once for the lifetime of the process.
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 55ms/step - accuracy: 0.2999 - loss: 2.1171 - val_a
ccuracy: 0.4375 - val_loss: 1.3322
Epoch 2/15
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.4528 - loss: 1.2018 - val_a
ccuracy: 0.5312 - val_loss: 1.1227
Epoch 3/15
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6831 - loss: 0.9298 - val_a
ccuracy: 0.6562 - val_loss: 0.8174
Epoch 4/15
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8175 - loss: 0.5760 - val_a
ccuracy: 0.6250 - val_loss: 0.7703
Epoch 5/15
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8898 - loss: 0.3115 - val_a
ccuracy: 0.6562 - val_loss: 0.6970
Epoch 6/15
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.9285 - loss: 0.2421 - val_a
ccuracy: 0.7188 - val_loss: 0.9824
Epoch 7/15
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9965 - loss: 0.1134 - val_a
ccuracy: 0.6250 - val_loss: 0.9379
Epoch 8/15
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9817 - loss: 0.0745 - val_ac
curacy: 0.7500 - val_loss: 0.9092
Epoch 9/15
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9965 - loss: 0.0307 - val_a
ccuracy: 0.6875 - val_loss: 1.2171
Epoch 10/15
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 1.0000 - loss: 0.0198 - val_a
ccuracy: 0.7188 - val_loss: 1.0783
Epoch 11/15
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 1.0000 - loss: 0.0098 - val_a
ccuracy: 0.6562 - val_loss: 1.2085
Epoch 12/15
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 0.0059 - val_ac
curacy: 0.7188 - val_loss: 1.2281
Epoch 13/15
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 1.0000 - loss: 0.0024 - val_a
ccuracy: 0.6875 - val_loss: 1.2855
Epoch 14/15
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.0023 - val_ac
curacy: 0.7188 - val_loss: 1.3567
Epoch 15/15
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 1.0000 - loss: 0.0018 - val_a
ccuracy: 0.7188 - val_loss: 1.3993
```

Evaluasi Model

In [ ]:

```python
import os
import tensorflow as tf

# Path disesuaikan
test_data_dir = '../dataset_split/test'
num_bad_files = 0

print(f"Mulai memeriksa file di dalam: {test_data_dir}")

# Looping ke semua file di dalam direktori dan subdirektori
for dirpath, dirnames, filenames in os.walk(test_data_dir):
    for filename in filenames:
        file_path = os.path.join(dirpath, filename)
```

```
        try:
            # Membaca dan mendekode file menggunakan TensorFlow
            raw_image = tf.io.read_file(file_path)
            tf.io.decode_image(raw_image)

        except tf.errors.InvalidArgumentError as e:
            # Jika TensorFlow gagal, file yang bermasalah
            print(f'❌ File bermasalah ditemukan: {file_path}')
            num_bad_files += 1

if num_bad_files == 0:
    print("✅ Semua file dalam folder test valid dan bisa dibaca TensorFlow!")
else:
    print(f"\nTotal ditemukan {num_bad_files} file bermasalah.")
```

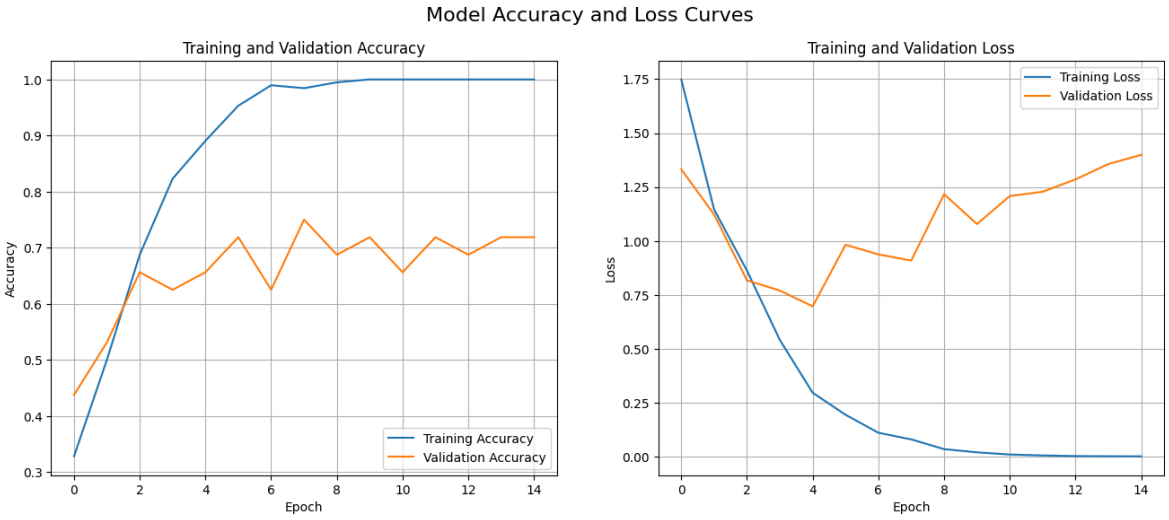
```
Mulai memeriksa file di dalam: ../dataset_split/test
✅ Semua file dalam folder test valid dan bisa dibaca TensorFlow!
```

In [14]:
```
test_loss, test_acc = model.evaluate(test_ds)
print(f'Test accuracy: {test_acc*100:.2f}%')
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 695ms/step - accuracy: 0.7738 - loss: 1.0453
Test accuracy: 78.57%
```

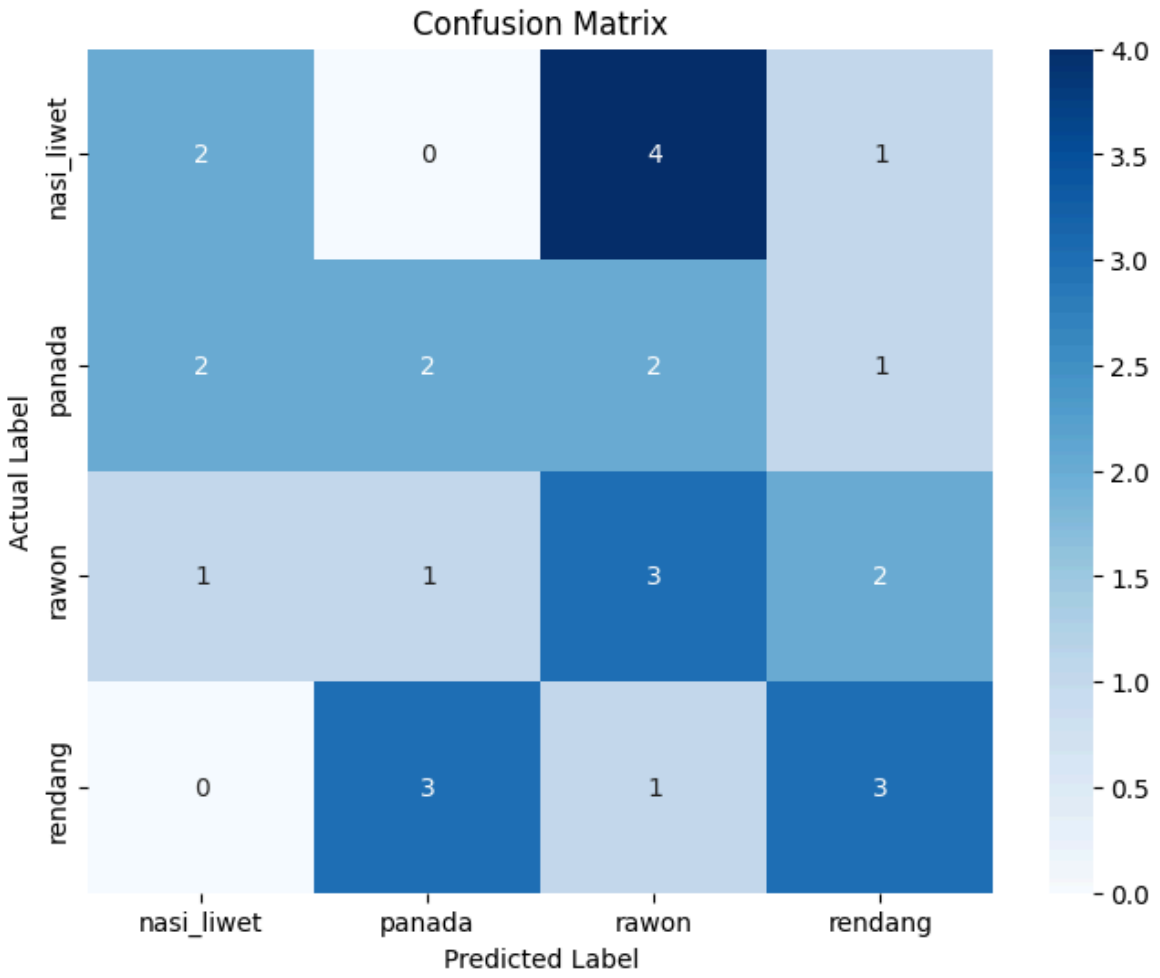
Visualisasi Hasil Training

In [16]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(len(acc))

# Membuat plot
plt.figure(figsize=(16, 6))

# Plot untuk Akurasi
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.grid(True)

# Plot untuk Loss
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.grid(True)

# Menampilkan grafik
plt.suptitle('Model Accuracy and Loss Curves', fontsize=16)
plt.show()
```



In [ ]:
```python
import seaborn as sns

train_dir = '../dataset_split/train' # Pastikan path ini sesuai
class_names = sorted(os.listdir(train_dir))
print(f"Class Names untuk Confusion Matrix: {class_names}")

# Mengambil label asli dari test_ds
y_true = np.concatenate([y for x, y in test_ds], axis=0)

# Melakukan prediksi pada seluruh data tes
y_pred_probs = model.predict(test_ds)
y_pred = np.argmax(y_pred_probs, axis=1)

# Menghitung confusion matrix
cm = tf.math.confusion_matrix(labels=y_true, predictions=y_pred).numpy()

# Membuat visualisasi confusion matrix menggunakan Seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names,
            yticklabels=class_names)

plt.title('Confusion Matrix')
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
plt.show()
```

```
Class Names untuk Confusion Matrix: ['nasi_liwet', 'panada', 'rawon', 'rendang']
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 85ms/step
```


Confusion Matrix
nasi_liwet
panada
rawon
rendang
Actual Label
2
0
4
1
2
2
2
1
1
1
3
2
0
3
1
3
nasi_liwet
panada
rawon
rendang
Predicted Label
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0

Import Library

In [1]:
```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout,
import matplotlib.pyplot as plt
import numpy as np
import os

# GPU Setting (optional tapi boleh sesuai latihan)
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
    except RuntimeError as e:
        print(e)
```

Data Load

In [2]:
```
TRAIN_DIR = "C:/Users/ASUS/Downloads/dataset_split/dataset_split/train"
VALIDATION_DIR = "C:/Users/ASUS/Downloads/dataset_split/dataset_split/validation"
TEST_DIR = "C:/Users/ASUS/Downloads/dataset_split/dataset_split/test"

IMG_SIZE = (224, 224)
NUM_CLASSES = 4
BATCH_SIZE = 16
EPOCHS = 30
```

Data Preprocessing

In [3]:
```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    zoom_range=0.2,
    horizontal_flip=True
)

val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_data = train_datagen.flow_from_directory(
    TRAIN_DIR,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical'
)

val_data = val_datagen.flow_from_directory(
    VALIDATION_DIR,
```

```
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical'
)

test_data = test_datagen.flow_from_directory(
    TEST_DIR,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=False
)

print("Label Mapping:", train_data.class_indices)
```

```
Found 240 images belonging to 4 classes.
Found 32 images belonging to 4 classes.
Found 28 images belonging to 4 classes.
Label Mapping: {'nasi_liwet': 0, 'panada': 1, 'rawon': 2, 'rendang': 3}
```

Arsitektur AlexNet

In [4]:

```
model = Sequential([
    # Blok Konvolusi 1
    # Input: 224x224x3
    Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_SIZE[0], IMG_SIZE[1], 3)
    MaxPooling2D((2, 2)),

    # Blok Konvolusi 2
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),

    # Blok Konvolusi 3
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),

    # Ratakan (Flatten) output untuk masuk ke bagian klasifikasi
    Flatten(),

    # Lapisan Klasifikasi (Fully Connected Layer)
    Dense(512, activation='relu'),
    Dropout(0.5), # Dropout sangat penting untuk mencegah overfitting pada data kec

    # Lapisan Output
    Dense(NUM_CLASSES, activation='softmax')
])

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

model.summary()
```

```
c:\Users\ASUS\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d (MaxPooling2D)	(None, 111, 111, 32)	0
conv2d_1 (Conv2D)	(None, 109, 109, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 54, 54, 64)	0
conv2d_2 (Conv2D)	(None, 52, 52, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 26, 26, 128)	0
flatten (Flatten)	(None, 86528)	0
dense (Dense)	(None, 512)	44,302,848
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 4)	2,052

Total params: 44,398,148 (169.37 MB)

Trainable params: 44,398,148 (169.37 MB)

Non-trainable params: 0 (0.00 B)

Training & Validation Model

In [5]:
```
history = model.fit(
    train_data,
    epochs=EPOCHS,
    validation_data=val_data
)

# Plot Accuracy & Loss
plt.figure(figsize=(12,4))

plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.legend()
plt.title('Accuracy')

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
```

```
plt.legend()
plt.title('Loss')

plt.show()
```

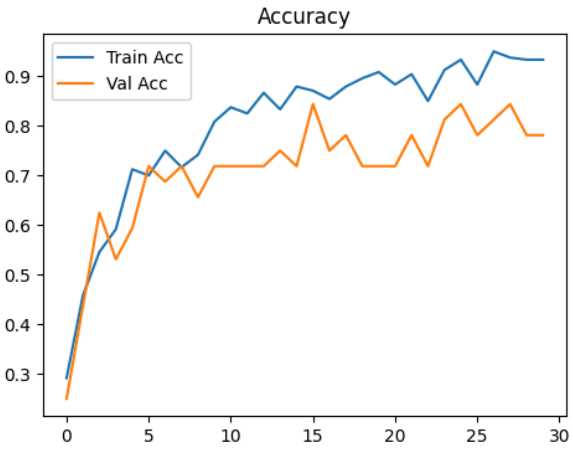
```
c:\Users\ASUS\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
```

```
Epoch 1/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 760ms/step - accuracy: 0.2187 - loss: 4.1604
```

```
c:\Users\ASUS\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 15s 821ms/step - accuracy: 0.2232 - loss: 4.0667 - val_ac
curacy: 0.2500 - val_loss: 1.3316
Epoch 2/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 743ms/step - accuracy: 0.3732 - loss: 1.2648 - val_ac
curacy: 0.4375 - val_loss: 1.2071
Epoch 3/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 757ms/step - accuracy: 0.5567 - loss: 1.0665 - val_ac
curacy: 0.6250 - val_loss: 0.9463
Epoch 4/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 766ms/step - accuracy: 0.5919 - loss: 0.9220 - val_ac
curacy: 0.5312 - val_loss: 0.9962
Epoch 5/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 761ms/step - accuracy: 0.7646 - loss: 0.7707 - val_ac
curacy: 0.5938 - val_loss: 0.9651
Epoch 6/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 767ms/step - accuracy: 0.7363 - loss: 0.6973 - val_ac
curacy: 0.7188 - val_loss: 0.6470
Epoch 7/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 777ms/step - accuracy: 0.7777 - loss: 0.6286 - val_ac
curacy: 0.6875 - val_loss: 0.9435
Epoch 8/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 771ms/step - accuracy: 0.7096 - loss: 0.6980 - val_ac
curacy: 0.7188 - val_loss: 0.7503
Epoch 9/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 761ms/step - accuracy: 0.7645 - loss: 0.6358 - val_ac
curacy: 0.6562 - val_loss: 0.8423
Epoch 10/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 784ms/step - accuracy: 0.8337 - loss: 0.4872 - val_ac
curacy: 0.7188 - val_loss: 0.7687
Epoch 11/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 775ms/step - accuracy: 0.8270 - loss: 0.4398 - val_ac
curacy: 0.7188 - val_loss: 0.7492
Epoch 12/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 746ms/step - accuracy: 0.8442 - loss: 0.3842 - val_ac
curacy: 0.7188 - val_loss: 0.5885
Epoch 13/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 762ms/step - accuracy: 0.8803 - loss: 0.3507 - val_ac
curacy: 0.7188 - val_loss: 0.7478
Epoch 14/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 767ms/step - accuracy: 0.8122 - loss: 0.4646 - val_ac
curacy: 0.7500 - val_loss: 0.6078
Epoch 15/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 750ms/step - accuracy: 0.8510 - loss: 0.3633 - val_ac
curacy: 0.7188 - val_loss: 0.9639
Epoch 16/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 773ms/step - accuracy: 0.8736 - loss: 0.3402 - val_ac
curacy: 0.8438 - val_loss: 0.4454
Epoch 17/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 732ms/step - accuracy: 0.8494 - loss: 0.3667 - val_ac
curacy: 0.7500 - val_loss: 0.5890
Epoch 18/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 751ms/step - accuracy: 0.8588 - loss: 0.4343 - val_ac
curacy: 0.7812 - val_loss: 0.4101
Epoch 19/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 763ms/step - accuracy: 0.9203 - loss: 0.1962 - val_ac
curacy: 0.7188 - val_loss: 0.6961
```

```
Epoch 20/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 740ms/step - accuracy: 0.9231 - loss: 0.1882 - val_ac
curacy: 0.7188 - val_loss: 0.5539
Epoch 21/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 740ms/step - accuracy: 0.9195 - loss: 0.2877 - val_ac
curacy: 0.7188 - val_loss: 0.7086
Epoch 22/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 757ms/step - accuracy: 0.8882 - loss: 0.3119 - val_ac
curacy: 0.7812 - val_loss: 0.7136
Epoch 23/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 755ms/step - accuracy: 0.9023 - loss: 0.3137 - val_ac
curacy: 0.7188 - val_loss: 0.6639
Epoch 24/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 738ms/step - accuracy: 0.9012 - loss: 0.3129 - val_ac
curacy: 0.8125 - val_loss: 0.5268
Epoch 25/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 771ms/step - accuracy: 0.9567 - loss: 0.1753 - val_ac
curacy: 0.8438 - val_loss: 0.4921
Epoch 26/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 741ms/step - accuracy: 0.8712 - loss: 0.2751 - val_ac
curacy: 0.7812 - val_loss: 0.5619
Epoch 27/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 769ms/step - accuracy: 0.9600 - loss: 0.1319 - val_ac
curacy: 0.8125 - val_loss: 0.6106
Epoch 28/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 766ms/step - accuracy: 0.9440 - loss: 0.1319 - val_ac
curacy: 0.8438 - val_loss: 0.5322
Epoch 29/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 770ms/step - accuracy: 0.9325 - loss: 0.1792 - val_ac
curacy: 0.7812 - val_loss: 0.7404
Epoch 30/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 786ms/step - accuracy: 0.9264 - loss: 0.1889 - val_ac
curacy: 0.7812 - val_loss: 0.5355
```

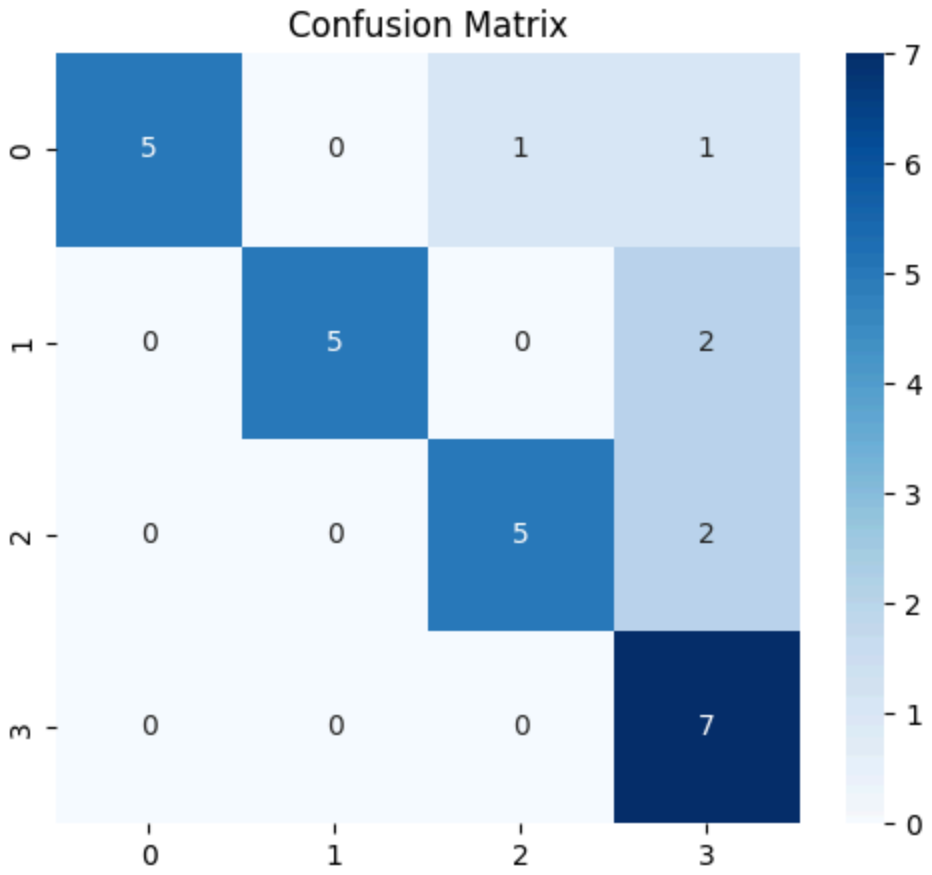


Evaluasi Model & Confusion Matrix

In [6]:
```
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

test_loss, test_acc = model.evaluate(test_data)
print("Test Accuracy:", test_acc)
```

```
y_pred = np.argmax(model.predict(test_data), axis=1)
y_true = test_data.classes

cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix")
plt.show()

print(classification_report(y_true, y_pred))
```

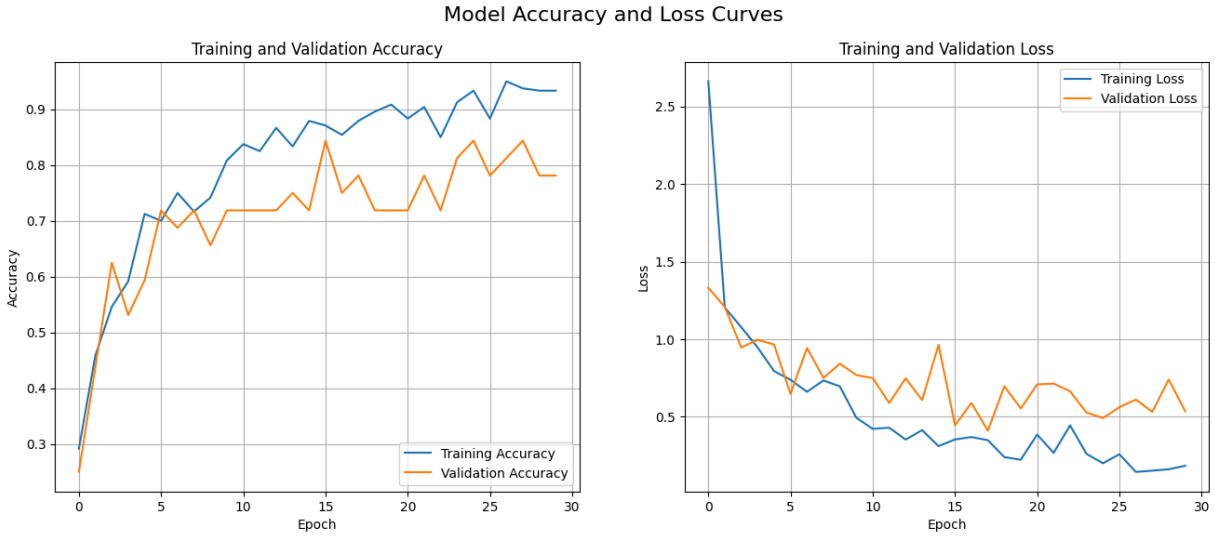
```
c:\Users\ASUS\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 234ms/step - accuracy: 0.7738 - loss: 1.0587
Test Accuracy: 0.7857142686843872
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 256ms/step
```



	precision	recall	f1-score	support
0	1.00	0.71	0.83	7
1	1.00	0.71	0.83	7
2	0.83	0.71	0.77	7
3	0.58	1.00	0.74	7
accuracy			0.79	28
macro avg	0.85	0.79	0.79	28
weighted avg	0.85	0.79	0.79	28

Visualisasi Akurasi & Loss

In [7]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(len(acc))

# Membuat plot
plt.figure(figsize=(16, 6))

# Plot untuk Akurasi
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.grid(True)

# Plot untuk Loss
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.grid(True)

# Menampilkan grafik
plt.suptitle('Model Accuracy and Loss Curves', fontsize=16)
plt.show()
```



Data Testing

In [8]:
```
test_loss, test_acc = model.evaluate(test_data)
print(f'Test accuracy: {test_acc*100:.2f}%')
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 237ms/step - accuracy: 0.7738 - loss: 1.0587
Test accuracy: 78.57%
```

Menyimpan Model Terbaik

In [9]:
```
model.save("UTS_DL_AlexNet.h5")
print("Model telah berhasil disimpan!")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
Model telah berhasil disimpan!
```

UTS_DL_MobileNet

October 27, 2025

0.1 Import Library

[1]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
```

```
2025-10-27 14:46:59.864807: I tensorflow/core/util/port.cc:153] oneDNN custom
operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn them
off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-10-27 14:46:59.886102: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
E0000 00:00:1761551219.909357 1222239 cuda_dnn.cc:8310] Unable to register cuDNN
factory: Attempting to register factory for plugin cuDNN when one has already
been registered
E0000 00:00:1761551219.916569 1222239 cuda_blas.cc:1418] Unable to register
cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has
already been registered
2025-10-27 14:46:59.941419: 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: AVX2 AVX512F AVX512_VNNI AVX512_BF16
AVX512_FP16 AVX_VNNI AMX_TILE AMX_INT8 AMX_BF16 FMA, in other operations,
rebuild TensorFlow with the appropriate compiler flags.
```

[2]:
```
gpus = tf.config.experimental.list_physical_devices('GPU')

for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
```

0.2 Load Dataset dan Data Preparation

[3]:
```
train_path = '../dataset/train'
valid_path = '../dataset/validation'
test_path = '../dataset/test'
```

[4]:
```
train_batches = ImageDataGenerator(
    preprocessing_function=tf.keras.applications.mobilenet.preprocess_input,
    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'
).flow_from_directory(
    directory = train_path, target_size = (150,150), batch_size = 4)
valid_batches = ImageDataGenerator(preprocessing_function=tf.keras.applications.
 ↪mobilenet.preprocess_input).flow_from_directory(
    directory = valid_path, target_size = (150,150), batch_size = 4)
test_batches = ImageDataGenerator(preprocessing_function=tf.keras.applications.
 ↪mobilenet.preprocess_input).flow_from_directory(
    directory = test_path, target_size = (150,150), batch_size = 4)

visual_batches = test_batches
```

```
Found 240 images belonging to 4 classes.
Found 32 images belonging to 4 classes.
Found 28 images belonging to 4 classes.
```

[5]:
```
assert train_batches.n == 240
assert valid_batches.n == 32
assert test_batches.n == 28
assert train_batches.num_classes == valid_batches.num_classes == test_batches.
 ↪num_classes == 4
```

0.3 Visualisasi Data

[6]:
```
display_datagen = ImageDataGenerator()

display_batches = display_datagen.flow_from_directory(
```

```
    directory=train_path,
    target_size=(150, 150),
    batch_size=4,
    class_mode='categorical',
    shuffle=True
)

def tampilkan_sampel_gambar(generator, jumlah=4):
    gambar, label = next(generator)

    plt.figure(figsize=(15, 5))
    for i in range(jumlah):
        plt.subplot(1, jumlah, i + 1)
        plt.imshow(gambar[i].astype('uint8'))

        nama_kelas = list(generator.class_indices.keys())[np.argmax(label[i])]
        plt.title(nama_kelas)
        plt.axis('off')
    plt.show()

print("Contoh Gambar dari Training Set:")
tampilkan_sampel_gambar(display_batches, 4)
```

```
Found 240 images belonging to 4 classes.
Contoh Gambar dari Training Set:
```

0.4 Arsitektur Model

[7]:
```
base_model = MobileNet(
    weights=None,
    include_top=False,
    input_shape=(150, 150, 3)
)
```

```
I0000 00:00:1761551224.352915 1222239 gpu_device.cc:2022] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 753 MB memory:  -> device: 0,
```

```
name: NVIDIA L40S, pci bus id: 0000:16:00.0, compute capability: 8.9
I0000 00:00:1761551224.355084 1222239 gpu_device.cc:2022] Created device
/job:localhost/replica:0/task:0/device:GPU:1 with 2531 MB memory:  -> device: 1,
name: NVIDIA L40S, pci bus id: 0000:be:00.0, compute capability: 8.9
```

[8]:
```
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dropout(0.3),
    Dense(4, activation='softmax')
])

model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
mobilenet_1.00_150 (Functional)	(None, 4, 4, 1024)	3,228,864
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1024)	0
dropout (Dropout)	(None, 1024)	0
dense (Dense)	(None, 4)	4,100

Total params: 3,232,964 (12.33 MB)

Trainable params: 3,211,076 (12.25 MB)

Non-trainable params: 21,888 (85.50 KB)

[9]:
```
model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

0.5 Model Training

[10]:
```
history = model.fit(
    train_batches,
    validation_data=valid_batches,
    epochs=120,
)
```

```
Epoch 1/120
```

```
/opt/tljh/user/envs/dltf/lib/python3.12/site-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()
WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
I0000 00:00:1761551233.623621 1222738 service.cc:148] XLA service 0x7fe1d4015dc0
initialized for platform CUDA (this does not guarantee that XLA will be used).
Devices:
I0000 00:00:1761551233.623684 1222738 service.cc:156]   StreamExecutor device
(0): NVIDIA L40S, Compute Capability 8.9
I0000 00:00:1761551233.623691 1222738 service.cc:156]   StreamExecutor device
(1): NVIDIA L40S, Compute Capability 8.9
2025-10-27 14:47:13.965531: I
tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:268] disabling MLIR
crash reproducer, set env var `MLIR_CRASH_REPRODUCER_DIRECTORY` to enable.
I0000 00:00:1761551235.197857 1222738 cuda_dnn.cc:529] Loaded cuDNN version
91002
E0000 00:00:1761551240.682291 1222738 gpu_timer.cc:82] Delay kernel timed out:
measured time has sub-optimal accuracy. There may be a missing warmup execution,
please investigate in Nsight Systems.
E0000 00:00:1761551240.769317 1222738 gpu_timer.cc:82] Delay kernel timed out:
measured time has sub-optimal accuracy. There may be a missing warmup execution,
please investigate in Nsight Systems.
I0000 00:00:1761551249.153341 1222738 device_compiler.h:188] Compiled cluster
using XLA!  This line is logged at most once for the lifetime of the process.
```

```
60/60             31s 90ms/step -
accuracy: 0.3201 - loss: 1.6069 - val_accuracy: 0.2500 - val_loss: 1.5191
Epoch 2/120
60/60             2s 40ms/step -
accuracy: 0.4259 - loss: 1.5394 - val_accuracy: 0.2500 - val_loss: 1.6875
Epoch 3/120
60/60             2s 39ms/step -
accuracy: 0.3765 - loss: 1.4528 - val_accuracy: 0.2500 - val_loss: 2.1053
Epoch 4/120
```

```
60/60            3s 44ms/step -
accuracy: 0.4269 - loss: 1.4673 - val_accuracy: 0.2500 - val_loss: 2.2679
Epoch 5/120
60/60            2s 41ms/step -
accuracy: 0.5024 - loss: 1.2299 - val_accuracy: 0.2500 - val_loss: 2.6496
Epoch 6/120
60/60            3s 45ms/step -
accuracy: 0.4434 - loss: 1.2868 - val_accuracy: 0.2500 - val_loss: 2.3974
Epoch 7/120
60/60            3s 44ms/step -
accuracy: 0.4722 - loss: 1.1398 - val_accuracy: 0.2500 - val_loss: 2.8126
Epoch 8/120
60/60            2s 37ms/step -
accuracy: 0.5002 - loss: 1.1332 - val_accuracy: 0.2500 - val_loss: 2.5736
Epoch 9/120
60/60            2s 40ms/step -
accuracy: 0.5879 - loss: 1.1321 - val_accuracy: 0.2500 - val_loss: 2.8622
Epoch 10/120
60/60            3s 41ms/step -
accuracy: 0.5240 - loss: 1.1383 - val_accuracy: 0.2500 - val_loss: 3.2928
Epoch 11/120
60/60            3s 45ms/step -
accuracy: 0.5461 - loss: 1.1144 - val_accuracy: 0.2500 - val_loss: 2.6718
Epoch 12/120
60/60            2s 40ms/step -
accuracy: 0.5081 - loss: 1.1583 - val_accuracy: 0.4688 - val_loss: 1.0917
Epoch 13/120
60/60            3s 44ms/step -
accuracy: 0.5496 - loss: 1.0158 - val_accuracy: 0.4688 - val_loss: 0.9977
Epoch 14/120
60/60            3s 41ms/step -
accuracy: 0.6351 - loss: 0.9779 - val_accuracy: 0.6250 - val_loss: 0.9716
Epoch 15/120
60/60            3s 45ms/step -
accuracy: 0.5797 - loss: 1.0081 - val_accuracy: 0.4062 - val_loss: 2.3937
Epoch 16/120
60/60            3s 41ms/step -
accuracy: 0.5947 - loss: 0.9470 - val_accuracy: 0.4375 - val_loss: 3.0057
Epoch 17/120
60/60            3s 43ms/step -
accuracy: 0.6323 - loss: 0.9189 - val_accuracy: 0.5000 - val_loss: 0.9615
Epoch 18/120
60/60            3s 41ms/step -
accuracy: 0.6258 - loss: 0.9609 - val_accuracy: 0.5000 - val_loss: 1.2645
Epoch 19/120
60/60            3s 43ms/step -
accuracy: 0.6127 - loss: 1.0384 - val_accuracy: 0.3750 - val_loss: 3.4459
Epoch 20/120
```

```
60/60            2s 41ms/step -
accuracy: 0.6215 - loss: 0.8882 - val_accuracy: 0.5938 - val_loss: 1.1685
Epoch 21/120
60/60            2s 40ms/step -
accuracy: 0.5701 - loss: 1.0266 - val_accuracy: 0.4062 - val_loss: 2.7106
Epoch 22/120
60/60            3s 46ms/step -
accuracy: 0.5720 - loss: 1.0107 - val_accuracy: 0.4062 - val_loss: 3.9878
Epoch 23/120
60/60            3s 41ms/step -
accuracy: 0.6376 - loss: 0.9560 - val_accuracy: 0.4062 - val_loss: 2.0662
Epoch 24/120
60/60            3s 43ms/step -
accuracy: 0.6446 - loss: 0.8735 - val_accuracy: 0.4375 - val_loss: 3.7347
Epoch 25/120
60/60            2s 38ms/step -
accuracy: 0.5817 - loss: 0.9280 - val_accuracy: 0.6250 - val_loss: 1.2158
Epoch 26/120
60/60            3s 44ms/step -
accuracy: 0.6530 - loss: 0.8015 - val_accuracy: 0.4688 - val_loss: 1.9436
Epoch 27/120
60/60            3s 44ms/step -
accuracy: 0.7212 - loss: 0.8204 - val_accuracy: 0.3750 - val_loss: 3.8402
Epoch 28/120
60/60            3s 46ms/step -
accuracy: 0.6280 - loss: 1.0092 - val_accuracy: 0.6875 - val_loss: 0.6742
Epoch 29/120
60/60            2s 40ms/step -
accuracy: 0.6603 - loss: 0.8444 - val_accuracy: 0.5312 - val_loss: 1.4861
Epoch 30/120
60/60            3s 43ms/step -
accuracy: 0.6570 - loss: 0.8169 - val_accuracy: 0.8125 - val_loss: 0.5604
Epoch 31/120
60/60            3s 41ms/step -
accuracy: 0.5363 - loss: 1.0661 - val_accuracy: 0.5625 - val_loss: 1.6992
Epoch 32/120
60/60            3s 43ms/step -
accuracy: 0.7697 - loss: 0.6848 - val_accuracy: 0.6562 - val_loss: 0.8765
Epoch 33/120
60/60            3s 42ms/step -
accuracy: 0.6966 - loss: 0.8429 - val_accuracy: 0.6875 - val_loss: 1.2758
Epoch 34/120
60/60            3s 45ms/step -
accuracy: 0.6835 - loss: 0.7539 - val_accuracy: 0.4688 - val_loss: 1.8064
Epoch 35/120
60/60            3s 42ms/step -
accuracy: 0.6551 - loss: 0.9094 - val_accuracy: 0.7500 - val_loss: 0.7379
Epoch 36/120
```

```
60/60            3s 46ms/step -
accuracy: 0.6920 - loss: 0.7159 - val_accuracy: 0.5625 - val_loss: 0.9725
Epoch 37/120
60/60            2s 39ms/step -
accuracy: 0.6233 - loss: 0.9502 - val_accuracy: 0.5000 - val_loss: 1.3514
Epoch 38/120
60/60            3s 47ms/step -
accuracy: 0.7313 - loss: 0.7741 - val_accuracy: 0.4062 - val_loss: 1.8369
Epoch 39/120
60/60            2s 41ms/step -
accuracy: 0.6881 - loss: 0.8644 - val_accuracy: 0.5000 - val_loss: 1.5494
Epoch 40/120
60/60            2s 39ms/step -
accuracy: 0.7355 - loss: 0.7605 - val_accuracy: 0.7500 - val_loss: 0.9253
Epoch 41/120
60/60            2s 39ms/step -
accuracy: 0.7069 - loss: 0.7206 - val_accuracy: 0.4375 - val_loss: 2.4587
Epoch 42/120
60/60            2s 37ms/step -
accuracy: 0.6339 - loss: 0.8957 - val_accuracy: 0.5312 - val_loss: 1.5188
Epoch 43/120
60/60            3s 50ms/step -
accuracy: 0.7259 - loss: 0.7391 - val_accuracy: 0.4062 - val_loss: 1.8525
Epoch 44/120
60/60            2s 36ms/step -
accuracy: 0.7489 - loss: 0.7046 - val_accuracy: 0.5625 - val_loss: 1.4456
Epoch 45/120
60/60            3s 42ms/step -
accuracy: 0.7232 - loss: 0.7623 - val_accuracy: 0.5000 - val_loss: 1.8587
Epoch 46/120
60/60            2s 38ms/step -
accuracy: 0.7613 - loss: 0.6570 - val_accuracy: 0.6250 - val_loss: 1.7098
Epoch 47/120
60/60            2s 34ms/step -
accuracy: 0.6787 - loss: 0.8312 - val_accuracy: 0.6875 - val_loss: 0.9788
Epoch 48/120
60/60            3s 42ms/step -
accuracy: 0.6860 - loss: 0.8265 - val_accuracy: 0.7188 - val_loss: 0.8496
Epoch 49/120
60/60            3s 41ms/step -
accuracy: 0.7402 - loss: 0.6683 - val_accuracy: 0.4688 - val_loss: 2.2111
Epoch 50/120
60/60            3s 44ms/step -
accuracy: 0.6842 - loss: 0.8552 - val_accuracy: 0.6250 - val_loss: 1.1298
Epoch 51/120
60/60            3s 41ms/step -
accuracy: 0.7212 - loss: 0.7797 - val_accuracy: 0.5625 - val_loss: 1.4453
Epoch 52/120
```

```
60/60            2s 38ms/step -
accuracy: 0.7743 - loss: 0.6779 - val_accuracy: 0.5625 - val_loss: 2.1227
Epoch 53/120
60/60            2s 40ms/step -
accuracy: 0.7757 - loss: 0.7608 - val_accuracy: 0.7188 - val_loss: 0.8848
Epoch 54/120
60/60            3s 47ms/step -
accuracy: 0.7126 - loss: 0.7916 - val_accuracy: 0.5312 - val_loss: 1.7240
Epoch 55/120
60/60            3s 42ms/step -
accuracy: 0.7881 - loss: 0.6474 - val_accuracy: 0.6250 - val_loss: 1.5104
Epoch 56/120
60/60            3s 44ms/step -
accuracy: 0.7107 - loss: 0.6457 - val_accuracy: 0.5000 - val_loss: 3.2779
Epoch 57/120
60/60            2s 40ms/step -
accuracy: 0.7482 - loss: 0.6948 - val_accuracy: 0.6562 - val_loss: 0.9613
Epoch 58/120
60/60            2s 39ms/step -
accuracy: 0.6711 - loss: 0.7583 - val_accuracy: 0.6250 - val_loss: 1.2974
Epoch 59/120
60/60            3s 45ms/step -
accuracy: 0.8053 - loss: 0.5366 - val_accuracy: 0.4688 - val_loss: 1.4331
Epoch 60/120
60/60            2s 38ms/step -
accuracy: 0.7547 - loss: 0.6577 - val_accuracy: 0.5312 - val_loss: 1.9731
Epoch 61/120
60/60            2s 38ms/step -
accuracy: 0.6972 - loss: 0.6789 - val_accuracy: 0.5312 - val_loss: 1.2333
Epoch 62/120
60/60            3s 41ms/step -
accuracy: 0.6951 - loss: 0.8022 - val_accuracy: 0.6875 - val_loss: 0.8230
Epoch 63/120
60/60            2s 39ms/step -
accuracy: 0.7220 - loss: 0.8542 - val_accuracy: 0.7812 - val_loss: 0.5350
Epoch 64/120
60/60            2s 41ms/step -
accuracy: 0.7795 - loss: 0.5799 - val_accuracy: 0.7812 - val_loss: 1.0030
Epoch 65/120
60/60            2s 38ms/step -
accuracy: 0.7350 - loss: 0.6331 - val_accuracy: 0.6562 - val_loss: 0.8894
Epoch 66/120
60/60            3s 44ms/step -
accuracy: 0.7425 - loss: 0.6407 - val_accuracy: 0.6875 - val_loss: 1.6200
Epoch 67/120
60/60            2s 37ms/step -
accuracy: 0.7760 - loss: 0.5885 - val_accuracy: 0.5625 - val_loss: 1.5218
Epoch 68/120
```

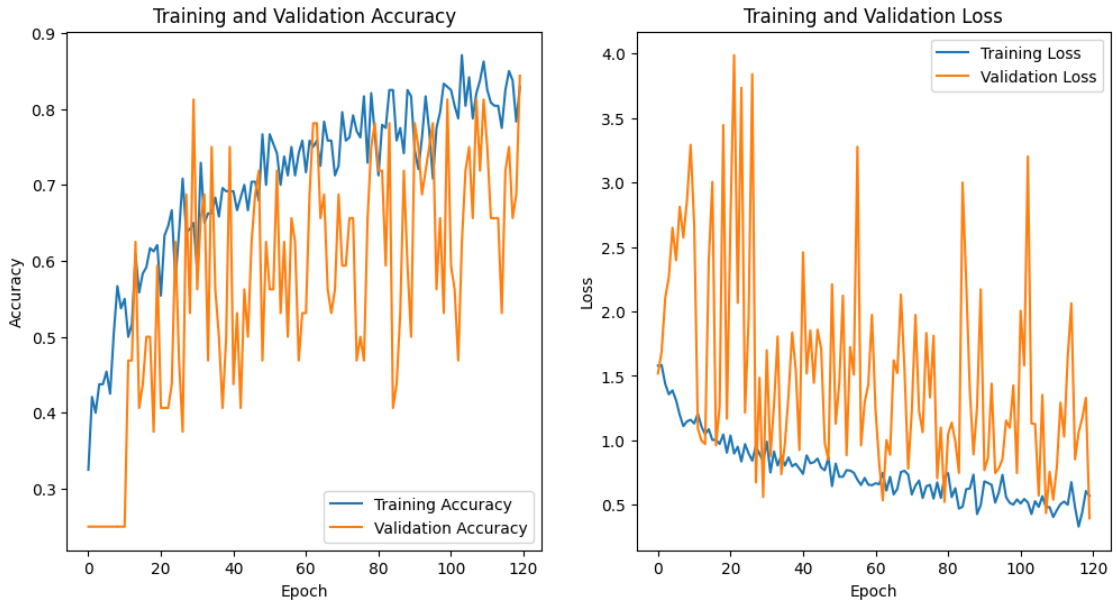
```
60/60            3s 42ms/step -
accuracy: 0.7576 - loss: 0.7427 - val_accuracy: 0.5312 - val_loss: 2.1317
Epoch 69/120
60/60            3s 42ms/step -
accuracy: 0.6810 - loss: 0.8191 - val_accuracy: 0.5625 - val_loss: 1.5407
Epoch 70/120
60/60            2s 41ms/step -
accuracy: 0.7300 - loss: 0.7216 - val_accuracy: 0.6875 - val_loss: 0.7810
Epoch 71/120
60/60            2s 39ms/step -
accuracy: 0.7859 - loss: 0.5887 - val_accuracy: 0.5938 - val_loss: 1.2294
Epoch 72/120
60/60            3s 44ms/step -
accuracy: 0.7910 - loss: 0.6566 - val_accuracy: 0.5938 - val_loss: 1.9734
Epoch 73/120
60/60            3s 44ms/step -
accuracy: 0.7886 - loss: 0.5902 - val_accuracy: 0.6562 - val_loss: 1.2264
Epoch 74/120
60/60            2s 40ms/step -
accuracy: 0.7821 - loss: 0.5693 - val_accuracy: 0.6562 - val_loss: 1.0635
Epoch 75/120
60/60            2s 35ms/step -
accuracy: 0.7739 - loss: 0.6355 - val_accuracy: 0.4688 - val_loss: 1.8331
Epoch 76/120
60/60            3s 46ms/step -
accuracy: 0.7402 - loss: 0.7083 - val_accuracy: 0.5000 - val_loss: 1.3335
Epoch 77/120
60/60            3s 44ms/step -
accuracy: 0.8023 - loss: 0.5299 - val_accuracy: 0.4688 - val_loss: 1.8113
Epoch 78/120
60/60            3s 41ms/step -
accuracy: 0.6827 - loss: 0.8240 - val_accuracy: 0.6562 - val_loss: 0.7062
Epoch 79/120
60/60            2s 38ms/step -
accuracy: 0.8381 - loss: 0.5174 - val_accuracy: 0.7500 - val_loss: 1.0995
Epoch 80/120
60/60            2s 39ms/step -
accuracy: 0.8161 - loss: 0.5717 - val_accuracy: 0.7812 - val_loss: 0.5225
Epoch 81/120
60/60            3s 42ms/step -
accuracy: 0.6725 - loss: 0.8088 - val_accuracy: 0.7188 - val_loss: 1.0483
Epoch 82/120
60/60            3s 42ms/step -
accuracy: 0.7823 - loss: 0.5343 - val_accuracy: 0.7188 - val_loss: 1.1381
Epoch 83/120
60/60            2s 38ms/step -
accuracy: 0.7680 - loss: 0.6843 - val_accuracy: 0.5938 - val_loss: 0.9911
Epoch 84/120
```

```
60/60            2s 38ms/step -
accuracy: 0.8609 - loss: 0.4282 - val_accuracy: 0.7812 - val_loss: 0.7474
Epoch 85/120
60/60            2s 37ms/step -
accuracy: 0.8252 - loss: 0.4214 - val_accuracy: 0.4062 - val_loss: 3.0003
Epoch 86/120
60/60            2s 38ms/step -
accuracy: 0.8018 - loss: 0.5301 - val_accuracy: 0.4375 - val_loss: 2.2275
Epoch 87/120
60/60            3s 44ms/step -
accuracy: 0.7737 - loss: 0.6158 - val_accuracy: 0.5312 - val_loss: 1.3911
Epoch 88/120
60/60            3s 41ms/step -
accuracy: 0.7519 - loss: 0.6388 - val_accuracy: 0.7188 - val_loss: 0.8914
Epoch 89/120
60/60            3s 46ms/step -
accuracy: 0.8425 - loss: 0.4196 - val_accuracy: 0.5938 - val_loss: 1.2594
Epoch 90/120
60/60            3s 42ms/step -
accuracy: 0.8312 - loss: 0.4663 - val_accuracy: 0.5000 - val_loss: 2.1716
Epoch 91/120
60/60            3s 45ms/step -
accuracy: 0.7708 - loss: 0.5796 - val_accuracy: 0.7812 - val_loss: 0.7694
Epoch 92/120
60/60            2s 41ms/step -
accuracy: 0.6926 - loss: 0.6839 - val_accuracy: 0.7500 - val_loss: 0.8630
Epoch 93/120
60/60            3s 41ms/step -
accuracy: 0.8011 - loss: 0.5390 - val_accuracy: 0.6875 - val_loss: 1.4407
Epoch 94/120
60/60            3s 43ms/step -
accuracy: 0.8108 - loss: 0.5566 - val_accuracy: 0.7188 - val_loss: 0.7463
Epoch 95/120
60/60            3s 46ms/step -
accuracy: 0.7515 - loss: 0.6054 - val_accuracy: 0.7500 - val_loss: 0.7843
Epoch 96/120
60/60            2s 40ms/step -
accuracy: 0.7127 - loss: 0.6356 - val_accuracy: 0.7812 - val_loss: 0.8552
Epoch 97/120
60/60            3s 44ms/step -
accuracy: 0.7663 - loss: 0.6135 - val_accuracy: 0.5625 - val_loss: 1.1543
Epoch 98/120
60/60            3s 42ms/step -
accuracy: 0.7663 - loss: 0.6432 - val_accuracy: 0.6562 - val_loss: 1.0983
Epoch 99/120
60/60            3s 46ms/step -
accuracy: 0.8771 - loss: 0.4355 - val_accuracy: 0.5312 - val_loss: 1.4265
Epoch 100/120
```

```
60/60            2s 40ms/step -
accuracy: 0.8556 - loss: 0.4951 - val_accuracy: 0.8125 - val_loss: 0.7458
Epoch 101/120
60/60            3s 47ms/step -
accuracy: 0.8427 - loss: 0.4878 - val_accuracy: 0.5938 - val_loss: 2.0058
Epoch 102/120
60/60            3s 43ms/step -
accuracy: 0.8301 - loss: 0.4931 - val_accuracy: 0.5625 - val_loss: 1.5811
Epoch 103/120
60/60            2s 40ms/step -
accuracy: 0.8115 - loss: 0.4710 - val_accuracy: 0.4688 - val_loss: 3.2027
Epoch 104/120
60/60            3s 43ms/step -
accuracy: 0.8651 - loss: 0.4695 - val_accuracy: 0.6250 - val_loss: 1.1310
Epoch 105/120
60/60            2s 39ms/step -
accuracy: 0.7853 - loss: 0.5639 - val_accuracy: 0.7188 - val_loss: 1.1282
Epoch 106/120
60/60            3s 43ms/step -
accuracy: 0.8568 - loss: 0.4705 - val_accuracy: 0.7500 - val_loss: 0.5725
Epoch 107/120
60/60            3s 42ms/step -
accuracy: 0.7942 - loss: 0.5648 - val_accuracy: 0.6562 - val_loss: 1.3530
Epoch 108/120
60/60            2s 39ms/step -
accuracy: 0.8395 - loss: 0.4622 - val_accuracy: 0.8125 - val_loss: 0.4350
Epoch 109/120
60/60            3s 42ms/step -
accuracy: 0.8203 - loss: 0.5330 - val_accuracy: 0.7188 - val_loss: 0.7561
Epoch 110/120
60/60            2s 38ms/step -
accuracy: 0.8679 - loss: 0.3463 - val_accuracy: 0.8125 - val_loss: 0.5400
Epoch 111/120
60/60            3s 47ms/step -
accuracy: 0.8458 - loss: 0.4399 - val_accuracy: 0.7500 - val_loss: 0.7977
Epoch 112/120
60/60            3s 41ms/step -
accuracy: 0.8249 - loss: 0.4617 - val_accuracy: 0.6562 - val_loss: 1.2930
Epoch 113/120
60/60            3s 51ms/step -
accuracy: 0.8262 - loss: 0.5201 - val_accuracy: 0.6562 - val_loss: 1.0291
Epoch 114/120
60/60            2s 38ms/step -
accuracy: 0.7917 - loss: 0.5389 - val_accuracy: 0.6562 - val_loss: 1.6504
Epoch 115/120
60/60            3s 44ms/step -
accuracy: 0.7756 - loss: 0.5481 - val_accuracy: 0.5312 - val_loss: 2.0632
Epoch 116/120
```

```
60/60              2s 36ms/step -
accuracy: 0.7774 - loss: 0.5277 - val_accuracy: 0.7188 - val_loss: 0.8530
Epoch 117/120
60/60              2s 40ms/step -
accuracy: 0.8680 - loss: 0.2844 - val_accuracy: 0.7500 - val_loss: 1.0606
Epoch 118/120
60/60              2s 38ms/step -
accuracy: 0.8603 - loss: 0.4324 - val_accuracy: 0.6562 - val_loss: 1.1726
Epoch 119/120
60/60              3s 44ms/step -
accuracy: 0.7937 - loss: 0.6243 - val_accuracy: 0.6875 - val_loss: 1.3300
Epoch 120/120
60/60              2s 37ms/step -
accuracy: 0.8619 - loss: 0.5153 - val_accuracy: 0.8438 - val_loss: 0.3957
```

[11]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(len(acc))

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')

plt.show()
```



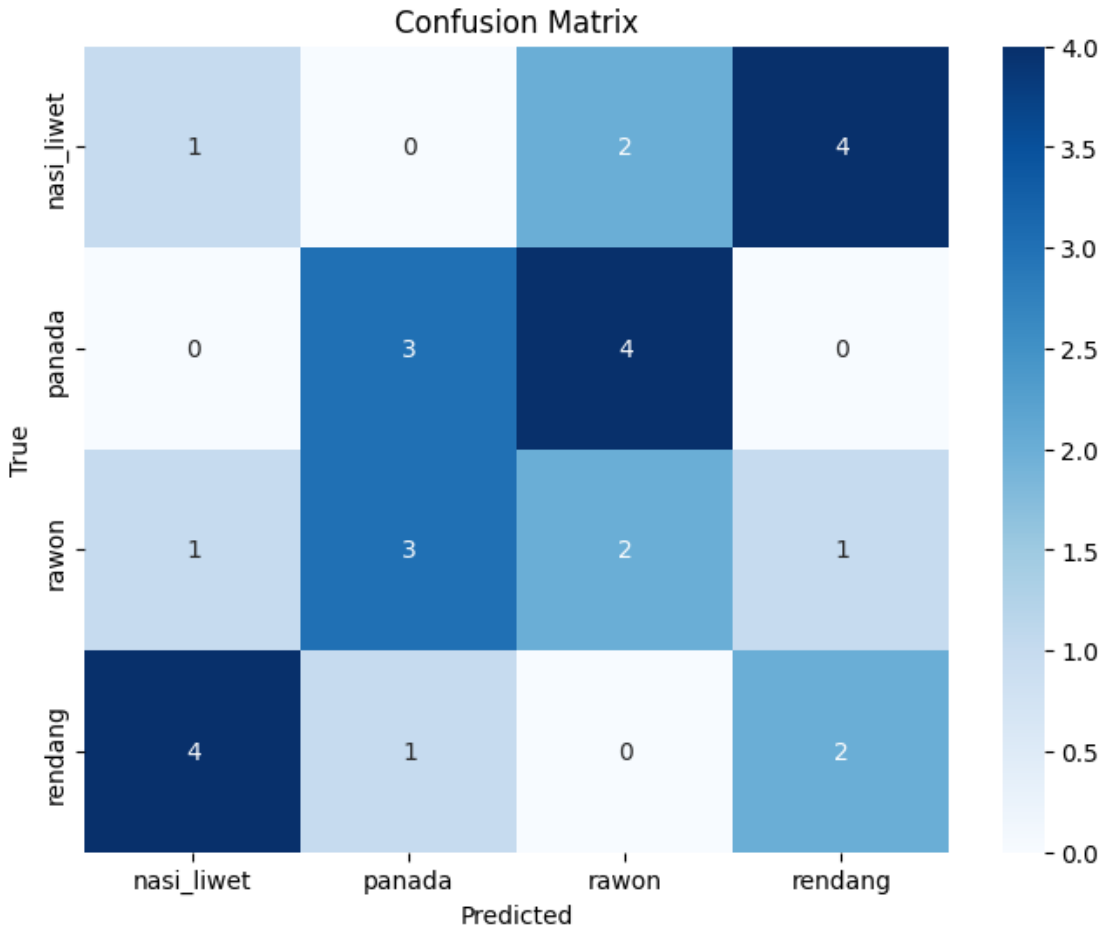
[12]:
```
loss, accuracy = model.evaluate(test_batches)

print(f"Loss pada data tes: {loss}")
print(f"Akurasi pada data tes: {accuracy * 100:.2f}%")
```

```
7/7             0s 47ms/step -
accuracy: 0.8670 - loss: 0.3578
Loss pada data tes: 0.3165914714336395
Akurasi pada data tes: 89.29%
```

0.6 Evaluasi Model

[13]:
```
predictions = model.predict(test_batches)
y_pred = np.argmax(predictions, axis=1)
y_true = test_batches.classes
class_labels = list(test_batches.class_indices.keys())

cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues', xticklabels=class_labels,␣
 ↪yticklabels=class_labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

```
7/7             3s 52ms/step
```



[14]:
```
model.save(" BestModel_MobileNet_GWS DL.h5")

print("Model telah berhasil disimpan!")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

Model telah berhasil disimpan!
```

[ ]:
```

```

```
import streamlit as st
import numpy as np
from PIL import Image
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout, GlobalAveragePooling2D

# ==============================================
# 1️⃣ Variabel dan Fungsi
# ==============================================

# Definisikan ukuran gambar dan nama kelas
# Pastikan keduanya SAMA PERSIS dengan yang ada di notebook training Anda
IMG_SIZE = (224, 224)
CLASS_NAMES = ['nasi_liwet', 'panada', 'rawon', 'rendang']

# Fungsi untuk membuat arsitektur model
def create_model(input_shape=(224,224,3), num_classes=4):
    # Arsitektur ini harus SAMA PERSIS dengan yang ada di notebook Anda
    model = Sequential([
        # Pastikan arsitektur ini cocok dengan model yang Anda latih
        Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_SIZE[0], IMG_SIZE[1], 3)),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        GlobalAveragePooling2D(),
        Dense(512, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])
    return model

# Fungsi untuk memuat model dan bobotnya
@st.cache_resource
def load_model_and_weights(weight_path):
    # Buat kerangka model yang "kosong"
    model = create_model(input_shape=(224,224,3), num_classes=4)
    # Muat "pengetahuan" (bobot) ke dalam kerangka tersebut
    model.load_weights(weight_path)
    return model
```

```
# Fungsi untuk prediksi gambar
def predict_image(model, img):
    # Ukuran gambar diubah agar sesuai dengan input model
    img = img.resize(IMG_SIZE)
    img_array = np.array(img)
    img_array = np.expand_dims(img_array, axis=0)
    # Pastikan preprocessing ini (/ 255.0) sama dengan saat training
    img_array = img_array / 255.0

    prediction = model.predict(img_array)
    return prediction

# ==============================================
# 2️⃣ Tampilan Utama Aplikasi Streamlit
# ==============================================
def main():
    st.title("Klasifikasi Makanan: Nasi Liwet, Panada, Rawon, Rendang")
    uploaded_file = st.file_uploader("Upload gambar makanan Anda", type=["jpg", "jpeg", "png"])

    if uploaded_file is not None:
        image = Image.open(uploaded_file).convert("RGB")
        st.image(image, caption="Gambar yang diupload", use_container_width=True)

        # Ganti dengan path ke file bobot (.weights.h5) Anda di GitHub
        model_path = "model/BestModel_AlexNet_GWS_DL.weights.h5"

        try:
            model = load_model_and_weights(model_path)

            if st.button('Prediksi Gambar'):
                with st.spinner('Model sedang menganalisis...'):
                    prediction = predict_image(model, image)

                    st.write("### Hasil Prediksi:")
                    predicted_class = CLASS_NAMES[np.argmax(prediction)]
                    confidence = np.max(prediction) * 100
                    st.success(f"Prediksi: *{predicted_class}*")
                    st.info(f"Tingkat Keyakinan: *{confidence:.2f}%*")

        except Exception as e:
            st.error(f"Gagal memuat model atau melakukan prediksi: {e}")

if _name_ == "_main_":
    main()
```