A Report
On

Multimodal Gas Detection & Classification Model Building

Course: DEEP LEARNING-UCS761

Submitted to:
Dr. Sushma Jain

Submitted by:
Alisha (102203037)

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

November 2025

Acknowledgement

I would like to express my sincere gratitude to Dr. Sushma Jain for her continuous guidance, support, and encouragement throughout the development of this project. Her valuable insights and feedback helped me understand the concepts of deep learning more clearly and apply them effectively in this work.

I am also thankful to the Department of Computer Science and Engineering for providing the opportunity to work on this project as part of the course DEEP LEARNING (UCS761). This project allowed me to gain practical experience in model building, data processing, and deep learning techniques.

Finally, I would like to thank my classmates, friends, and family for their constant motivation and support during the completion of this project.

INDEX

1. Background and Summary of Existing Approaches

Gas detection plays an important role in industrial safety, environmental monitoring, and early warning systems. Traditional gas detection systems mainly rely on electronic gas sensors. These sensors change their electrical properties when they come in contact with different gases. Although these systems are widely used, they face limitations. Sensor readings for different gases can sometimes look similar, and environmental factors such as temperature or humidity can affect the results. Because of these issues, sensor-only approaches may struggle to accurately separate multiple gas types.

To improve accuracy, recent research has explored using image-based methods. Thermal cameras can capture the heat signatures produced by gases. These heat patterns are not always visible to the human eye but can be learned by deep learning models. Convolutional Neural Networks (CNNs) have proven effective in recognizing patterns in thermal images. They automatically learn features such as shapes, textures, and heat variations. However, image-only methods also have limitations. They may detect the shape of the gas cloud but cannot measure the chemical properties or concentration of the gas.

Because both sensor-only and image-only approaches have weaknesses, many existing studies now focus on multimodal methods. In multimodal systems, sensor data and thermal images are combined together. Sensor data gives accurate chemical readings, while images give visual patterns. This combination helps the model become more reliable and reduces misclassification. For example, if two gases produce similar sensor readings, the thermal image can help separate them. On the other hand, if two gases look similar in images, the sensor readings help distinguish them.

In this project, we follow the same multimodal approach. We build three models: a base CNN model using only images, a transfer learning model using MobileNetV2, and a hybrid model that combines both sensor and image data. These models help us compare performance and understand the strengths of each approach. The hybrid model aims to use the advantages of both sensor and image information for better accuracy and reliability.

2. Dataset Description

This project uses a multimodal gas detection dataset containing both sensor readings and thermal camera images. Each sample includes seven gas sensor values and one matching thermal image. The dataset contains 6400 samples.

1. Sensor Data (CSV File)

The CSV file contains numerical readings from seven gas sensors, the gas label, and the matching image name.

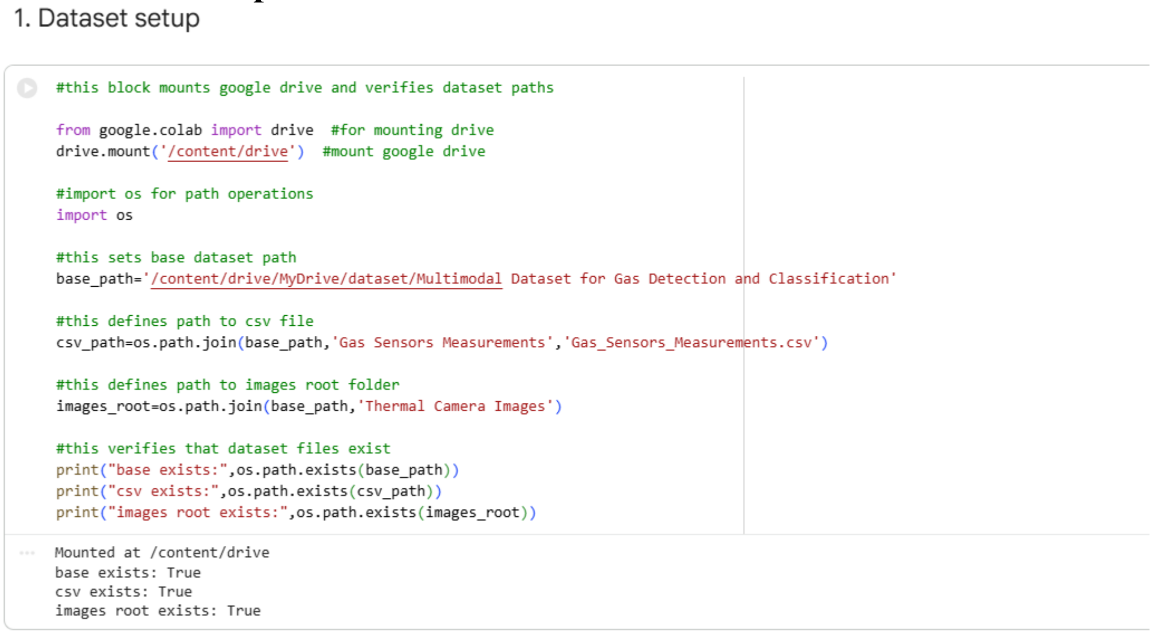
Column	Type	Description
Serial Number	Integer	Unique ID for each sample
MQ2	Integer	Sensor reading for MQ2 gas sensor
MQ3	Integer	Sensor reading for MQ3 gas sensor
MQ5	Integer	Sensor reading for MQ5 gas sensor
MQ6	Integer	Sensor reading for MQ6 gas sensor
MQ7	Integer	Sensor reading for MQ7 gas sensor
MQ8	Integer	Sensor reading for MQ8 gas sensor
MQ135	Integer	Sensor reading for MQ135 gas sensor
Gas	Category	Class label (NoGas, Smoke, Perfume, Mixture)
Corresponding Image Name	String	Name of the thermal image file

2. Thermal Camera Images

The images are thermal infrared images stored in four folders: NoGas, Smoke, Perfume, and Mixture. Each image corresponds to one row in the CSV file. The images reflect heat patterns formed by the gases.

3. Methodology

3.1 Dataset Setup



1. Dataset setup

```
#this block mounts google drive and verifies dataset paths

from google.colab import drive  #for mounting drive
drive.mount('/content/drive')  #mount google drive

#import os for path operations
import os

#this sets base dataset path
base_path='/content/drive/MyDrive/dataset/Multimodal Dataset for Gas Detection and Classification'

#this defines path to csv file
csv_path=os.path.join(base_path,'Gas Sensors Measurements','Gas_Sensors_Measurements.csv')

#this defines path to images root folder
images_root=os.path.join(base_path,'Thermal Camera Images')

#this verifies that dataset files exist
print("base exists:",os.path.exists(base_path))
print("csv exists:",os.path.exists(csv_path))
print("images root exists:",os.path.exists(images_root))
```

```
Mounted at /content/drive
base exists: True
csv exists: True
images root exists: True
```

Explanation:

This step involves connecting Google Drive to Google Colab to access the dataset files. Since all dataset folders and files were uploaded to Google Drive, the first task was to mount the drive using the drive.mount() function. After mounting, the code defines three important paths:

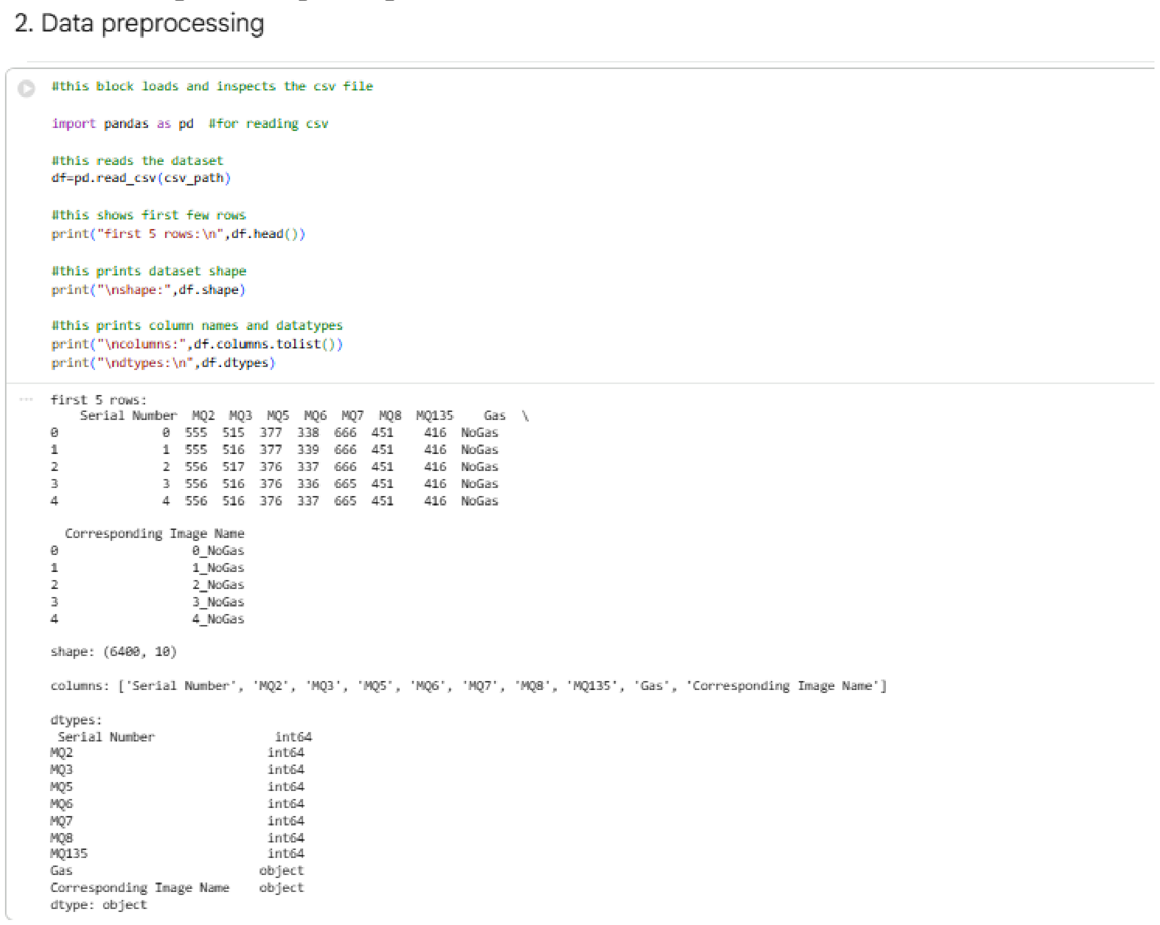
1. The **base dataset folder**
2. The **CSV file path**, which contains numerical sensor readings
3. The **image root folder**, which contains thermal images organized in four sub-folders (NoGas, Smoke, Perfume, Mixture)

The code then verifies whether all required paths exist on the drive using os.path.exists(). If all paths return "True", it confirms that the dataset is correctly located and ready to be processed. This setup ensures that the next steps (preprocessing and model building) can run without file-path errors.

3.2 Data Pre-processing

Data pre-processing is one of the most important steps of this project because our dataset is multimodal. Each sample contains **sensor readings** and a **thermal image**, so both must be cleaned, verified, and aligned. This section explains how we loaded the CSV, checked image availability, pre-processed the data, encoded labels, and created the final train/validation/test splits.

3.2.1 Reading and Inspecting the CSV File

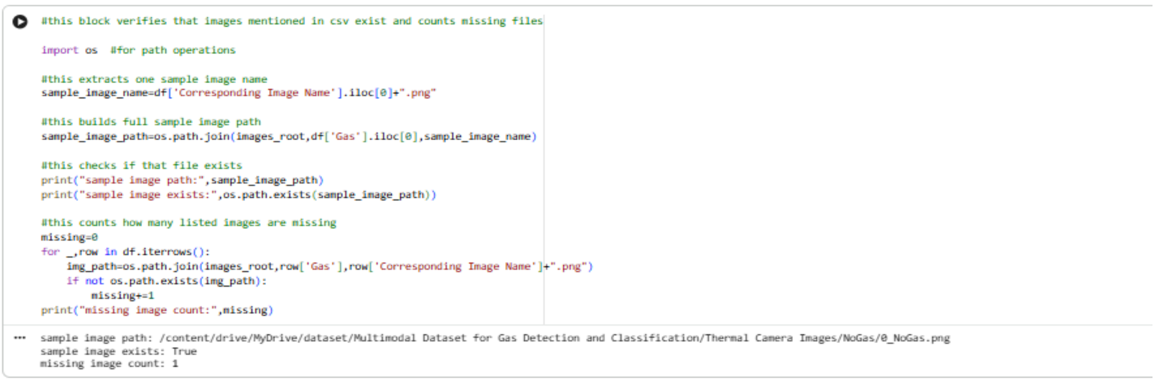

2. Data preprocessing

```
#this block loads and inspects the csv file

import pandas as pd  #for reading csv

#this reads the dataset
df=pd.read_csv(csv_path)

#this shows first few rows
print("first 5 rows:\n",df.head())

#this prints dataset shape
print("\nshape:",df.shape)

#this prints column names and datatypes
print("\ncolumns:",df.columns.tolist())
print("\ndtypes:\n",df.dtypes)
```

```
first 5 rows:
    Serial Number  MQ2  MQ3  MQ5  MQ6  MQ7  MQ8  MQ135    Gas  \
0              0  555  515  377  338  666  451    416  NoGas
1              1  555  516  377  339  666  451    416  NoGas
2              2  556  517  376  337  666  451    416  NoGas
3              3  556  516  376  336  665  451    416  NoGas
4              4  556  516  376  337  665  451    416  NoGas

  Corresponding Image Name
0                  0_NoGas
1                  1_NoGas
2                  2_NoGas
3                  3_NoGas
4                  4_NoGas

shape: (6400, 10)

columns: ['Serial Number', 'MQ2', 'MQ3', 'MQ5', 'MQ6', 'MQ7', 'MQ8', 'MQ135', 'Gas', 'Corresponding Image Name']

dtypes:
 Serial Number                int64
MQ2                          int64
MQ3                          int64
MQ5                          int64
MQ6                          int64
MQ7                          int64
MQ8                          int64
MQ135                        int64
Gas                         object
Corresponding Image Name    object
dtype: object
```

This block reads the CSV file using *pandas* and prints the basic information needed to understand the dataset. The first few rows are displayed to confirm correct loading, followed by the shape of the dataset (6400 rows and 10 columns). The column names and their respective data types are printed to verify that the sensor readings are integers, while the gas label and image names are stored as strings. This initial inspection ensures that the dataset is correctly structured and ready for further preprocessing.

3.2.2 Verifying Image Availability



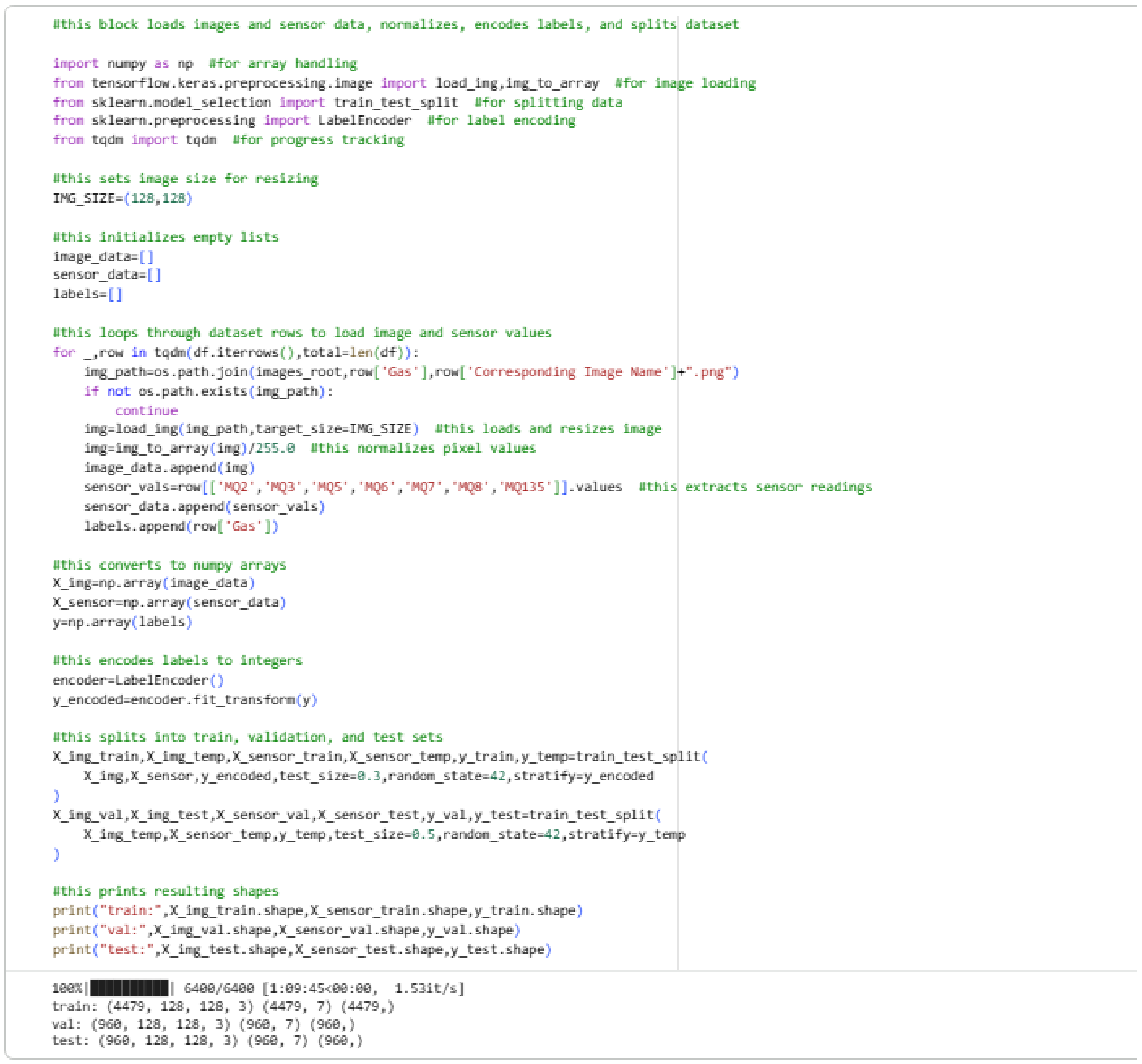
```
#this block verifies that images mentioned in csv exist and counts missing files

import os  #for path operations

#this extracts one sample image name
sample_image_name=df['Corresponding Image Name'].iloc[0]+".png"

#this builds full sample image path
sample_image_path=os.path.join(images_root,df['Gas'].iloc[0],sample_image_name)

#this checks if that file exists
print("sample image path:",sample_image_path)
print("sample image exists:",os.path.exists(sample_image_path))

#this counts how many listed images are missing
missing=0
for _,row in df.iterrows():
    img_path=os.path.join(images_root,row['Gas'],row['Corresponding Image Name']+".png")
    if not os.path.exists(img_path):
        missing+=1
print("missing image count:",missing)
```

```
sample image path: /content/drive/MyDrive/dataset/Multimodal Dataset for Gas Detection and Classification/Thermal Camera Images/NoGas/0_NoGas.png
sample image exists: True
missing image count: 1
```

This block checks whether the image files listed in the CSV actually exist inside the thermal image folders.

A sample image path is constructed to confirm the folder structure, and then a loop iterates through all rows to count how many image files are missing. This step is important because mismatches between the CSV entries and the actual images can cause errors later during training. If any images are missing, they can be handled before proceeding.

3.2.3 Loading, Normalizing, Encoding and Splitting the Data



```
#this block loads images and sensor data, normalizes, encodes labels, and splits dataset

import numpy as np  #for array handling
from tensorflow.keras.preprocessing.image import load_img,img_to_array  #for image loading
from sklearn.model_selection import train_test_split  #for splitting data
from sklearn.preprocessing import LabelEncoder  #for label encoding
from tqdm import tqdm  #for progress tracking

#this sets image size for resizing
IMG_SIZE=(128,128)

#this initializes empty lists
image_data=[]
sensor_data=[]
labels=[]

#this loops through dataset rows to load image and sensor values
for _,row in tqdm(df.iterrows(),total=len(df)):
    img_path=os.path.join(images_root,row['Gas'],row['Corresponding Image Name']+".png")
    if not os.path.exists(img_path):
        continue
    img=load_img(img_path,target_size=IMG_SIZE)  #this loads and resizes image
    img=img_to_array(img)/255.0  #this normalizes pixel values
    image_data.append(img)
    sensor_vals=row[['MQ2','MQ3','MQ5','MQ6','MQ7','MQ8','MQ135']].values  #this extracts sensor readings
    sensor_data.append(sensor_vals)
    labels.append(row['Gas'])

#this converts to numpy arrays
X_img=np.array(image_data)
X_sensor=np.array(sensor_data)
y=np.array(labels)

#this encodes labels to integers
encoder=LabelEncoder()
y_encoded=encoder.fit_transform(y)

#this splits into train, validation, and test sets
X_img_train,X_img_temp,X_sensor_train,X_sensor_temp,y_train,y_temp=train_test_split(
    X_img,X_sensor,y_encoded,test_size=0.3,random_state=42,stratify=y_encoded
)
X_img_val,X_img_test,X_sensor_val,X_sensor_test,y_val,y_test=train_test_split(
    X_img_temp,X_sensor_temp,y_temp,test_size=0.5,random_state=42,stratify=y_temp
)

#this prints resulting shapes
print("train:",X_img_train.shape,X_sensor_train.shape,y_train.shape)
print("val:",X_img_val.shape,X_sensor_val.shape,y_val.shape)
print("test:",X_img_test.shape,X_sensor_test.shape,y_test.shape)
```

```
100%|██████████| 6400/6400 [1:09:45<00:00,  1.53it/s]
train: (4479, 128, 128, 3) (4479, 7) (4479,)
val: (960, 128, 128, 3) (960, 7) (960,)
test: (960, 128, 128, 3) (960, 7) (960,)
```

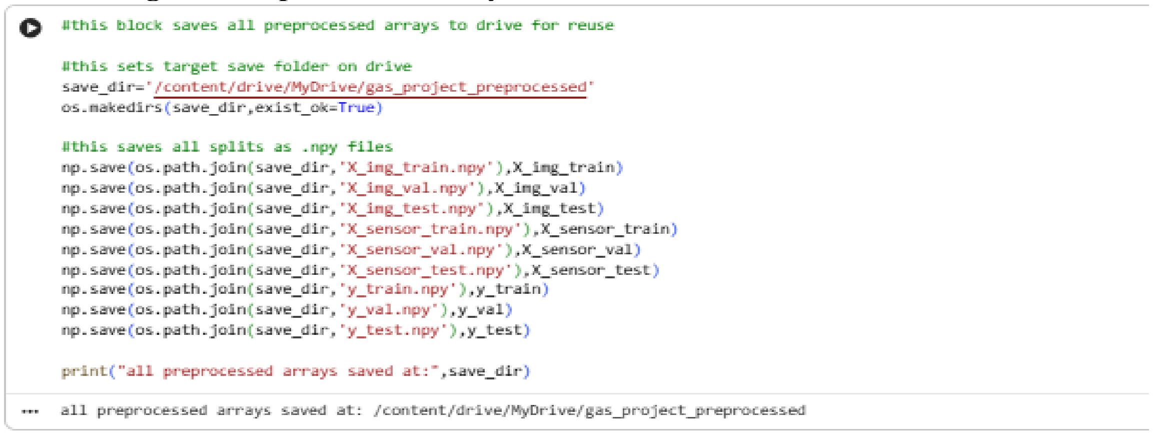
This block performs the main preprocessing steps:

- **Image Loading:** Each image is loaded and resized to **128×128** pixels for uniformity.
- **Normalization:** All pixel values are divided by 255.0 so they fall between 0 and 1, which helps the neural network train more effectively.
- **Sensor Extraction:** The seven MQ sensor readings are collected into a separate array.
- **Label Encoding:** Gas labels (e.g., NoGas, Smoke) are converted into numeric form for model training.

- **Train/Validation/Test Split:** The dataset is divided into 70% training, 15% validation, and 15% testing using stratified splitting to maintain class balance.

This ensures that both image data and sensor data are clean, normalized, and ready for training.

3.2.4 Saving the Pre-processed Arrays



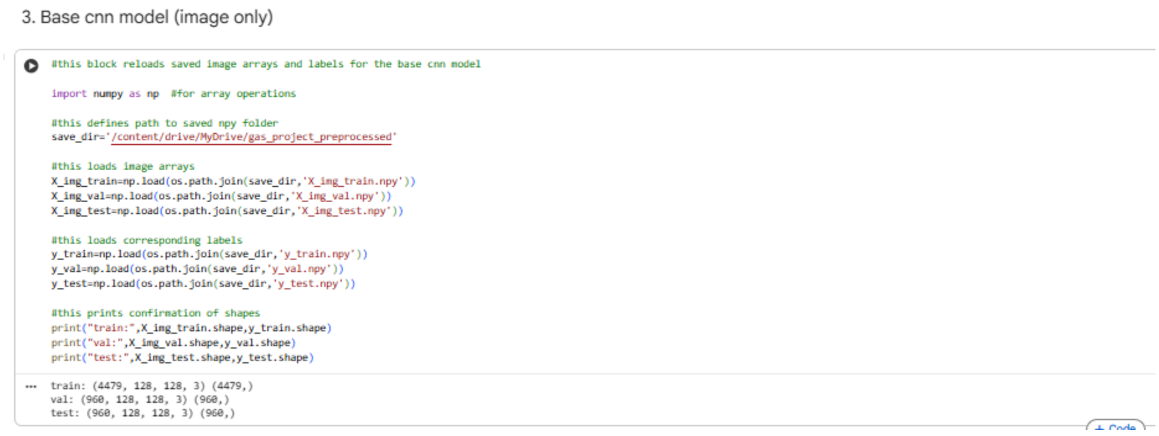
```
#this block saves all preprocessed arrays to drive for reuse

#this sets target save folder on drive
save_dir='/content/drive/MyDrive/gas_project_preprocessed'
os.makedirs(save_dir,exist_ok=True)

#this saves all splits as .npy files
np.save(os.path.join(save_dir,'X_img_train.npy'),X_img_train)
np.save(os.path.join(save_dir,'X_img_val.npy'),X_img_val)
np.save(os.path.join(save_dir,'X_img_test.npy'),X_img_test)
np.save(os.path.join(save_dir,'X_sensor_train.npy'),X_sensor_train)
np.save(os.path.join(save_dir,'X_sensor_val.npy'),X_sensor_val)
np.save(os.path.join(save_dir,'X_sensor_test.npy'),X_sensor_test)
np.save(os.path.join(save_dir,'y_train.npy'),y_train)
np.save(os.path.join(save_dir,'y_val.npy'),y_val)
np.save(os.path.join(save_dir,'y_test.npy'),y_test)

print("all preprocessed arrays saved at:",save_dir)
```

```
all preprocessed arrays saved at: /content/drive/MyDrive/gas_project_preprocessed
```

This block saves all preprocessed arrays (X_img, X_sensor, and y) into .npy files stored in Google Drive.
Saving these files means the entire preprocessing pipeline does not need to be repeated each time.
This makes the training process faster and ensures that the same consistent data is used whenever the models are reloaded.

3.3 Base CNN Model (Image Only)

3.3.1 Reloading Pre-processed Image Data



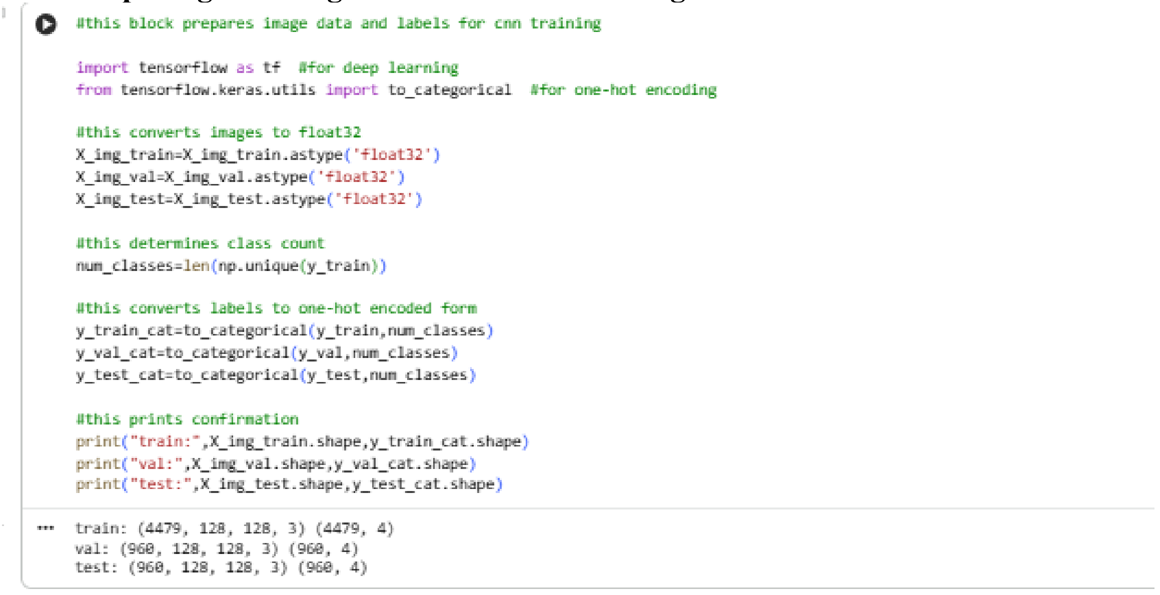
3. Base cnn model (image only)

```
#this block reloads saved image arrays and labels for the base cnn model

import numpy as np  #for array operations

#this defines path to saved npy folder
save_dir='/content/drive/MyDrive/gas_project_preprocessed'

#this loads image arrays
X_img_train=np.load(os.path.join(save_dir,'X_img_train.npy'))
X_img_val=np.load(os.path.join(save_dir,'X_img_val.npy'))
X_img_test=np.load(os.path.join(save_dir,'X_img_test.npy'))

#this loads corresponding labels
y_train=np.load(os.path.join(save_dir,'y_train.npy'))
y_val=np.load(os.path.join(save_dir,'y_val.npy'))
y_test=np.load(os.path.join(save_dir,'y_test.npy'))

#this prints confirmation of shapes
print("train:",X_img_train.shape,y_train.shape)
print("val:",X_img_val.shape,y_val.shape)
print("test:",X_img_test.shape,y_test.shape)
```

```
train: (4479, 128, 128, 3) (4479,)
val: (960, 128, 128, 3) (960,)
test: (960, 128, 128, 3) (960,)
```

This code block loads the saved NumPy arrays for the image-only CNN model. Since preprocessing was already done earlier, we directly import the training, validation, and test image datasets along with their labels. Storing and reloading data in .npy format makes the workflow faster and ensures that the same clean, consistent data is used during training. This step is important because it keeps the preprocessing separate from the model training stage, avoids accidental reprocessing, and ensures reproducibility.

3.3.2 Preparing the Image Data for CNN Training

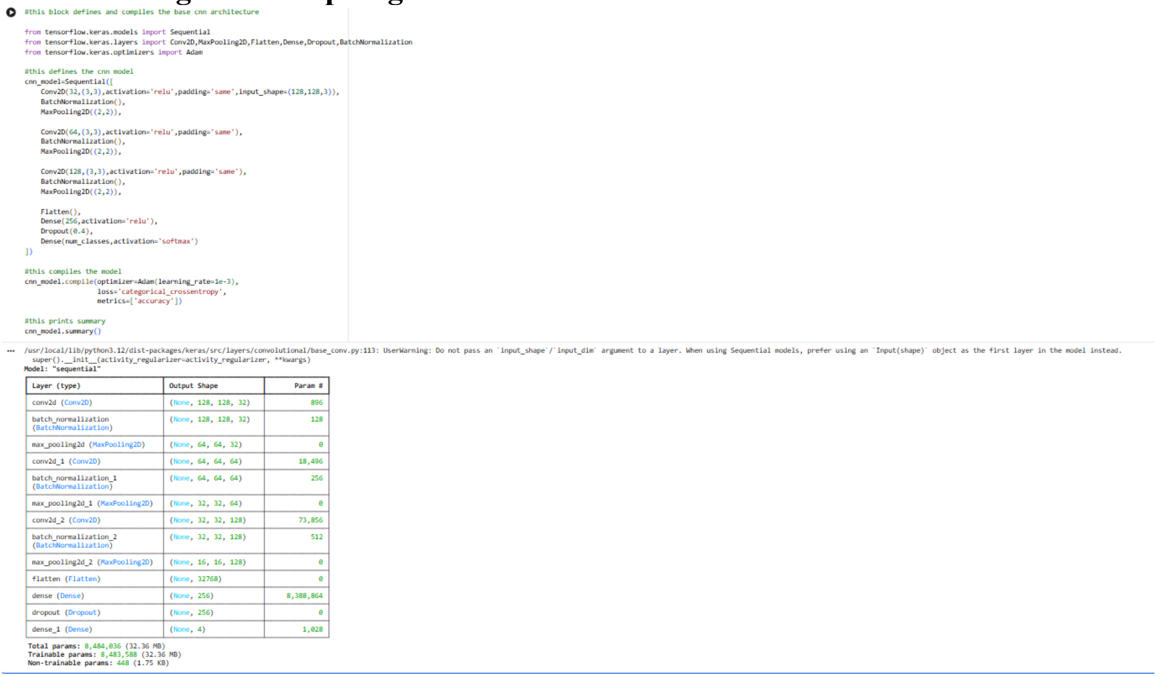


```python
#this block prepares image data and labels for cnn training

import tensorflow as tf  #for deep learning
from tensorflow.keras.utils import to_categorical  #for one-hot encoding

#this converts images to float32
X_img_train=X_img_train.astype('float32')
X_img_val=X_img_val.astype('float32')
X_img_test=X_img_test.astype('float32')

#this determines class count
num_classes=len(np.unique(y_train))

#this converts labels to one-hot encoded form
y_train_cat=to_categorical(y_train,num_classes)
y_val_cat=to_categorical(y_val,num_classes)
y_test_cat=to_categorical(y_test,num_classes)

#this prints confirmation
print("train:",X_img_train.shape,y_train_cat.shape)
print("val:",X_img_val.shape,y_val_cat.shape)
print("test:",X_img_test.shape,y_test_cat.shape)
```

```
train: (4479, 128, 128, 3) (4479, 4)
val: (960, 128, 128, 3) (960, 4)
test: (960, 128, 128, 3) (960, 4)
```

In this block, the image arrays are converted to float32 so that they are compatible with TensorFlow operations. The class labels are converted into **one-hot encoded vectors**, which means each class is represented as a binary vector (e.g., NoGas → [1,0,0,0]). One-hot encoding is required because the CNN uses a softmax output layer, which expects labels in this format for multi-class classification. The shapes printed at the end confirm that both the images and labels are correctly formatted for training.

3.3.3 Defining and Compiling the Base CNN Architecture



```python
#this block defines and compiles the base cnn architecture

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D,MaxPooling2D,Flatten,Dense,Dropout,BatchNormalization
from tensorflow.keras.optimizers import Adam

#this defines the cnn model
cnn_model=Sequential([
    Conv2D(32,(3,3),activation='relu',padding='same',input_shape=(128,128,3)),
    BatchNormalization(),
    MaxPooling2D((2,2)),

    Conv2D(64,(3,3),activation='relu',padding='same'),
    BatchNormalization(),
    MaxPooling2D((2,2)),

    Conv2D(128,(3,3),activation='relu',padding='same'),
    BatchNormalization(),
    MaxPooling2D((2,2)),

    Flatten(),
    Dense(256,activation='relu'),
    Dropout(0.4),
    Dense(num_classes,activation='softmax')
])

#this compiles the model
cnn_model.compile(optimizer=Adam(learning_rate=1e-3),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

#this prints summary
cnn_model.summary()
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: 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, 128, 128, 32)   │           896 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization             │ (None, 128, 128, 32)   │           128 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d (MaxPooling2D)    │ (None, 64, 64, 32)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_1 (Conv2D)               │ (None, 64, 64, 64)     │        18,496 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_1           │ (None, 64, 64, 64)     │           256 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_1 (MaxPooling2D)  │ (None, 32, 32, 64)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_2 (Conv2D)               │ (None, 32, 32, 128)    │        73,856 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_2           │ (None, 32, 32, 128)    │           512 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_2 (MaxPooling2D)  │ (None, 16, 16, 128)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten (Flatten)               │ (None, 32768)          │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                   │ (None, 256)            │     8,388,864 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout (Dropout)               │ (None, 256)            │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 4)              │         1,028 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 8,484,036 (32.36 MB)
 Trainable params: 8,483,588 (32.36 MB)
 Non-trainable params: 448 (1.75 KB)
```

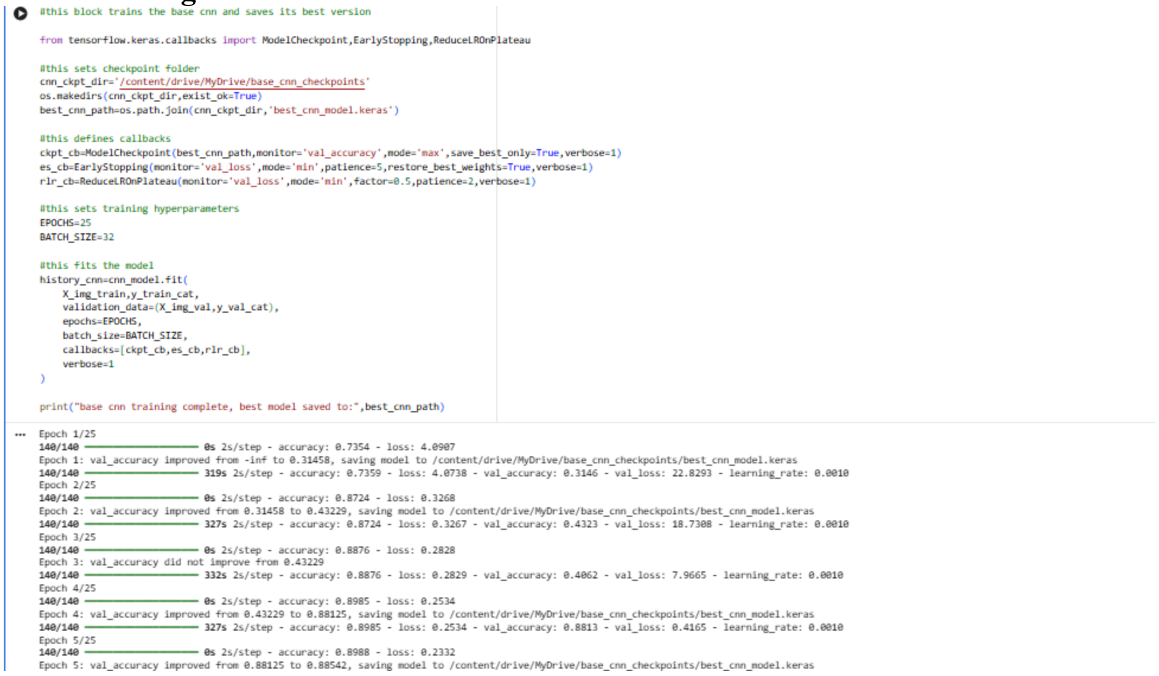
Here, a simple Convolutional Neural Network (CNN) is defined using stacked convolution layers followed by pooling layers.

- **Conv2D layers** extract image features such as edges, shapes, and textures.

- **BatchNormalization** stabilizes training and makes learning faster.
- **MaxPooling** reduces the spatial size and helps the network learn important patterns.
- **Flatten + Dense layers** convert extracted features into classification decisions.
- **Dropout** prevents overfitting.

The model is compiled using the **Adam optimizer**, **categorical cross-entropy loss**, and **accuracy** as the evaluation metric. The printed summary shows the number of parameters and the structure of each layer.

3.3.4 Training the Base CNN



```python
#this block trains the base cnn and saves its best version

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

#this sets checkpoint folder
cnn_ckpt_dir='/content/drive/MyDrive/base_cnn_checkpoints'
os.makedirs(cnn_ckpt_dir,exist_ok=True)
best_cnn_path=os.path.join(cnn_ckpt_dir,'best_cnn_model.keras')

#this defines callbacks
ckpt_cb=ModelCheckpoint(best_cnn_path,monitor='val_accuracy',mode='max',save_best_only=True,verbose=1)
es_cb=EarlyStopping(monitor='val_loss',mode='min',patience=5,restore_best_weights=True,verbose=1)
rlr_cb=ReduceLROnPlateau(monitor='val_loss',mode='min',factor=0.5,patience=2,verbose=1)

#this sets training hyperparameters
EPOCHS=25
BATCH_SIZE=32

#this fits the model
history_cnn=cnn_model.fit(
    X_img_train,y_train_cat,
    validation_data=(X_img_val,y_val_cat),
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    callbacks=[ckpt_cb,es_cb,rlr_cb],
    verbose=1
)

print("base cnn training complete, best model saved to:",best_cnn_path)
```

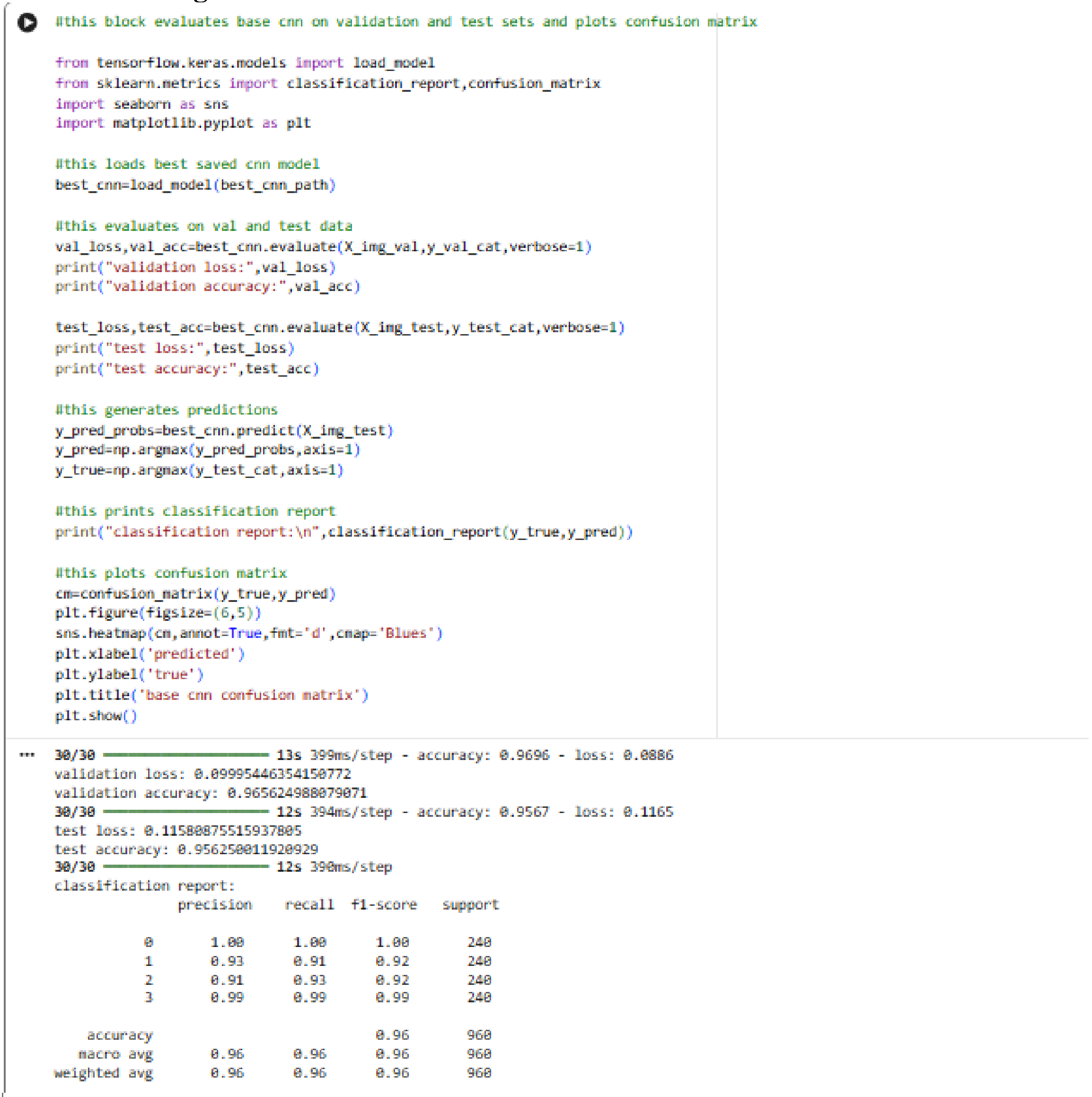
```
Epoch 1/25
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.7354 - loss: 4.0907
Epoch 1: val_accuracy improved from -inf to 0.31458, saving model to /content/drive/MyDrive/base_cnn_checkpoints/best_cnn_model.keras
140/140 ━━━━━━━━━━━━━━━━━━━━ 319s 2s/step - accuracy: 0.7359 - loss: 4.0738 - val_accuracy: 0.3146 - val_loss: 22.8293 - learning_rate: 0.0010
Epoch 2/25
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.8724 - loss: 0.3268
Epoch 2: val_accuracy improved from 0.31458 to 0.43229, saving model to /content/drive/MyDrive/base_cnn_checkpoints/best_cnn_model.keras
140/140 ━━━━━━━━━━━━━━━━━━━━ 327s 2s/step - accuracy: 0.8724 - loss: 0.3267 - val_accuracy: 0.4323 - val_loss: 18.7308 - learning_rate: 0.0010
Epoch 3/25
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.8876 - loss: 0.2828
Epoch 3: val_accuracy did not improve from 0.43229
140/140 ━━━━━━━━━━━━━━━━━━━━ 332s 2s/step - accuracy: 0.8876 - loss: 0.2829 - val_accuracy: 0.4062 - val_loss: 7.9665 - learning_rate: 0.0010
Epoch 4/25
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.8985 - loss: 0.2534
Epoch 4: val_accuracy improved from 0.43229 to 0.88125, saving model to /content/drive/MyDrive/base_cnn_checkpoints/best_cnn_model.keras
140/140 ━━━━━━━━━━━━━━━━━━━━ 327s 2s/step - accuracy: 0.8985 - loss: 0.2534 - val_accuracy: 0.8813 - val_loss: 0.4165 - learning_rate: 0.0010
Epoch 5/25
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.8988 - loss: 0.2332
Epoch 5: val_accuracy improved from 0.88125 to 0.88542, saving model to /content/drive/MyDrive/base_cnn_checkpoints/best_cnn_model.keras
```

This step trains the CNN on the training dataset. Three important callbacks are used:

- **ModelCheckpoint** saves the model that achieves the best validation accuracy.
- **EarlyStopping** automatically stops training if the model stops improving, preventing overfitting.
- **ReduceLROnPlateau** lowers the learning rate when progress slows down, helping the model learn better.

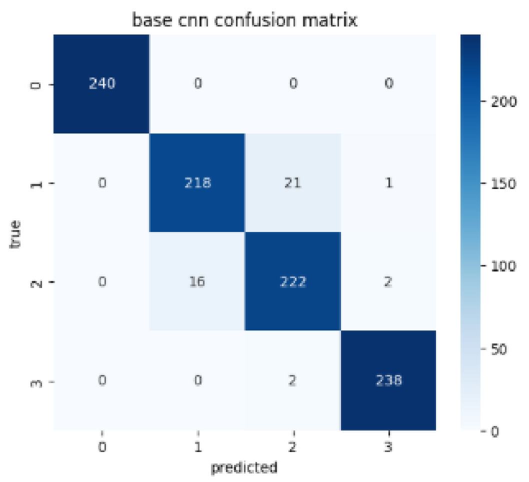
A batch size of 32 and 25 epochs are used. During training, improvements in validation accuracy are tracked. The best-performing model is saved for later evaluation.

3.3.5 Evaluating the Base CNN + Confusion Matrix



```
#this block evaluates base cnn on validation and test sets and plots confusion matrix

from tensorflow.keras.models import load_model
from sklearn.metrics import classification_report,confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

#this loads best saved cnn model
best_cnn=load_model(best_cnn_path)

#this evaluates on val and test data
val_loss,val_acc=best_cnn.evaluate(X_img_val,y_val_cat,verbose=1)
print("validation loss:",val_loss)
print("validation accuracy:",val_acc)

test_loss,test_acc=best_cnn.evaluate(X_img_test,y_test_cat,verbose=1)
print("test loss:",test_loss)
print("test accuracy:",test_acc)

#this generates predictions
y_pred_probs=best_cnn.predict(X_img_test)
y_pred=np.argmax(y_pred_probs,axis=1)
y_true=np.argmax(y_test_cat,axis=1)

#this prints classification report
print("classification report:\n",classification_report(y_true,y_pred))

#this plots confusion matrix
cm=confusion_matrix(y_true,y_pred)
plt.figure(figsize=(6,5))
sns.heatmap(cm,annot=True,fmt='d',cmap='Blues')
plt.xlabel('predicted')
plt.ylabel('true')
plt.title('base cnn confusion matrix')
plt.show()
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 13s 399ms/step - accuracy: 0.9696 - loss: 0.0886
validation loss: 0.09995446354150772
validation accuracy: 0.965624988079071
30/30 ━━━━━━━━━━━━━━━━━━━━ 12s 394ms/step - accuracy: 0.9567 - loss: 0.1165
test loss: 0.11580875515937805
test accuracy: 0.956250011920929
30/30 ━━━━━━━━━━━━━━━━━━━━ 12s 390ms/step
classification report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       240
           1       0.93      0.91      0.92       240
           2       0.91      0.93      0.92       240
           3       0.99      0.99      0.99       240

    accuracy                           0.96       960
   macro avg       0.96      0.96      0.96       960
weighted avg       0.96      0.96      0.96       960
```

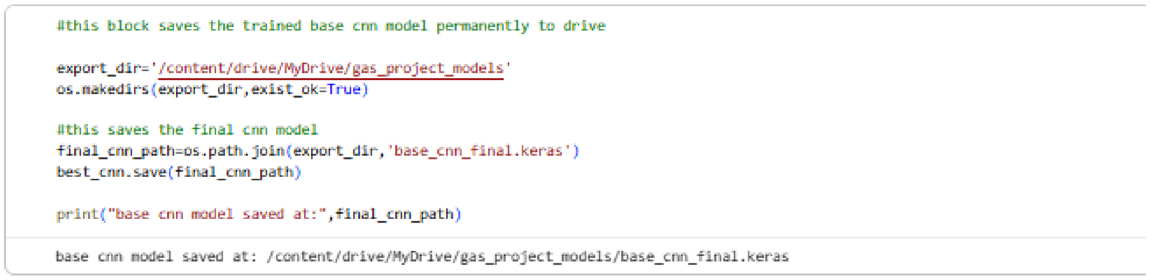


This block loads the saved best model and evaluates it on validation and test sets. The accuracy and loss values give a direct measure of model performance. Predictions are generated for all test images and compared with true labels to compute:

- **Classification report** (precision, recall, F1-score for each class)
- **Confusion matrix** (visual chart of correct vs. incorrect predictions)

The confusion matrix helps identify which gas classes are classified well and where errors occur. Overall, this section confirms the effectiveness of the base CNN before moving to transfer learning and hybrid modelling.

3.3.5 Saving the Final Base CNN Model



```
#this block saves the trained base cnn model permanently to drive

export_dir='/content/drive/MyDrive/gas_project_models'
os.makedirs(export_dir,exist_ok=True)

#this saves the final cnn model
final_cnn_path=os.path.join(export_dir,'base_cnn_final.keras')
best_cnn.save(final_cnn_path)

print("base cnn model saved at:",final_cnn_path)
```

```
base cnn model saved at: /content/drive/MyDrive/gas_project_models/base_cnn_final.keras
```

In this step, the final trained version of the Base CNN model is saved permanently so that it can be reused later without retraining. First, a new folder is created in Google Drive to store all project models. Then, the best CNN model (selected during training using the validation accuracy) is saved inside this folder in .keras format. Saving the model ensures that it can be easily loaded again for evaluation, comparison with other models, or deployment, without repeating the entire training process. This also keeps the project organized by separating the model files from the dataset and preprocessing outputs.

3.3.6 Base CNN Model — Hyperparameters

While training the base CNN model, I tested a few different hyperparameter values to check which ones give the most stable training.
I first tried:

- Learning rate: **0.01** → model accuracy jumped too much and became unstable
- Learning rate: **0.0005** → model learned too slowly
- Batch sizes: **16** → too slow,
- Batch size: **64** → training was unstable

After trying these variations, the final values that worked the best were:
Final hyperparameters used:

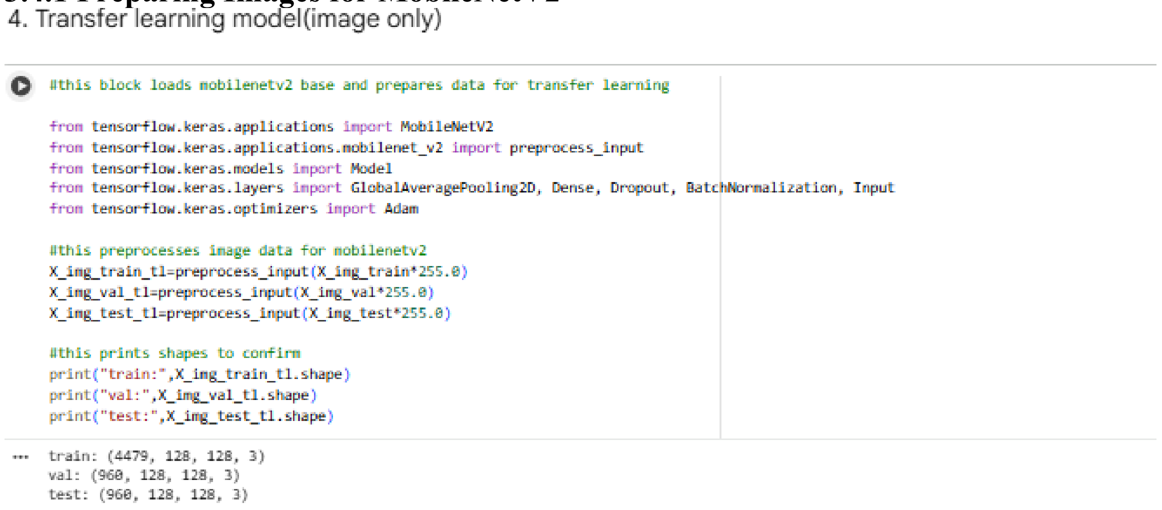
- **Learning Rate: 0.001**
- **Batch Size: 32**
- **Epochs: 25**
- **Optimizer:** Adam
- **Loss Function:** Categorical Cross-Entropy
- **Callbacks:** Early Stopping, Model Checkpoint, ReduceLROnPlateau

These settings gave smooth accuracy improvement and the best results for the base CNN.

3.4 Transfer Learning Model (Image Only)

This section describes the second model of our project, which uses **transfer learning** with the MobileNetV2 architecture. Transfer learning allows us to use a model that has already been trained on a large and general dataset (ImageNet) and apply its learned visual features to our gas-classification problem. Instead of training a CNN from scratch, we reuse the strong feature extraction ability of MobileNetV2 and add our own classification layers on top. This helps the model achieve higher accuracy even with a relatively small dataset.

3.4.1 Preparing Images for MobileNetV2



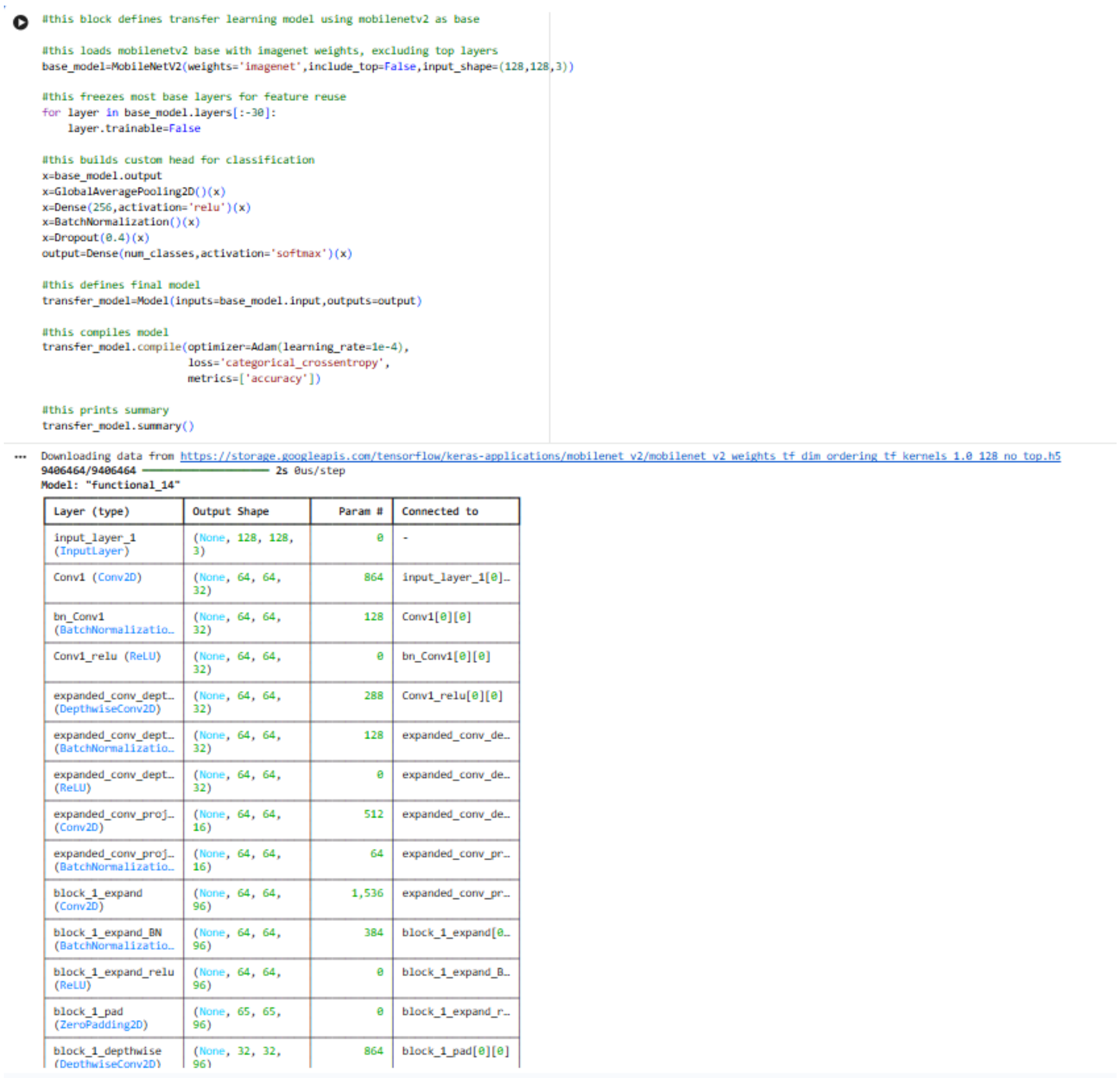
4. Transfer learning model(image only)

```
#this block loads mobilenetv2 base and prepares data for transfer learning

from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout, BatchNormalization, Input
from tensorflow.keras.optimizers import Adam

#this preprocesses image data for mobilenetv2
X_img_train_tl=preprocess_input(X_img_train*255.0)
X_img_val_tl=preprocess_input(X_img_val*255.0)
X_img_test_tl=preprocess_input(X_img_test*255.0)

#this prints shapes to confirm
print("train:",X_img_train_tl.shape)
print("val:",X_img_val_tl.shape)
print("test:",X_img_test_tl.shape)
```

```
train: (4479, 128, 128, 3)
val: (960, 128, 128, 3)
test: (960, 128, 128, 3)
```

This block imports MobileNetV2's preprocessing function and applies it to the training, validation, and test images. MobileNetV2 expects images in a specific normalized format, so preprocess_input() is used to convert our images correctly. We then verify the shapes of the processed datasets to ensure everything is ready for model training.

3.4.2 Building the MobileNetV2-Based Model

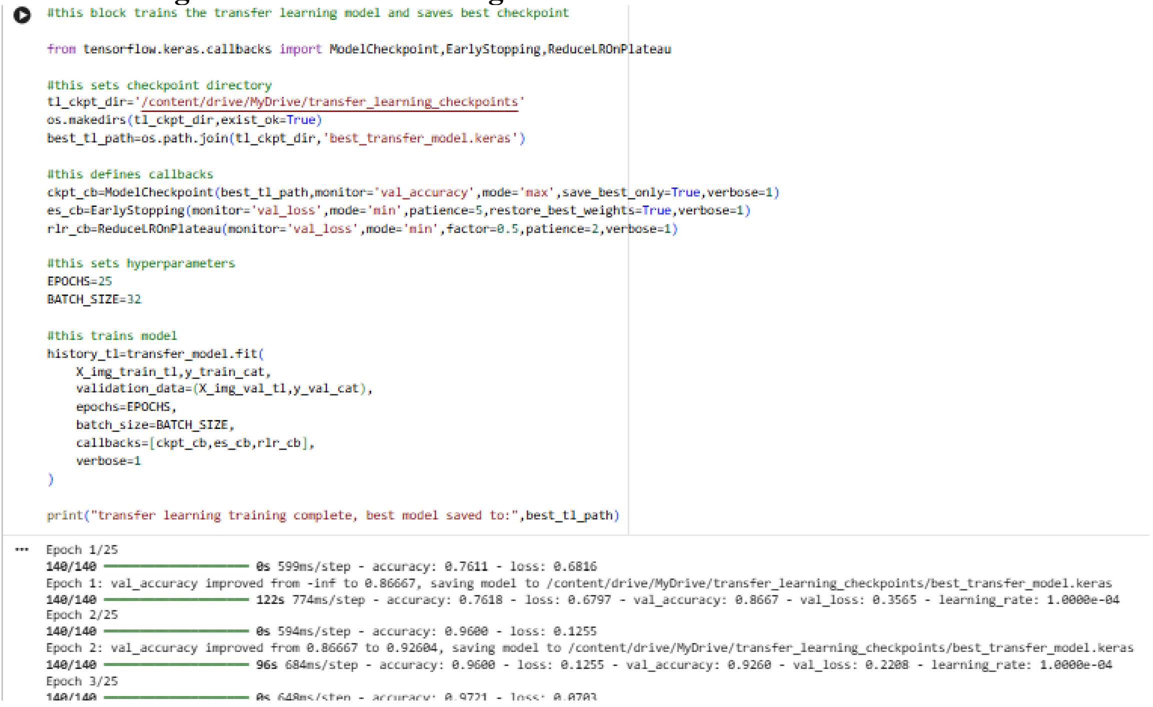


```python
#this block defines transfer learning model using mobilenetv2 as base

#this loads mobilenetv2 base with imagenet weights, excluding top layers
base_model=MobileNetV2(weights='imagenet',include_top=False,input_shape=(128,128,3))

#this freezes most base layers for feature reuse
for layer in base_model.layers[:-30]:
    layer.trainable=False

#this builds custom head for classification
x=base_model.output
x=GlobalAveragePooling2D()(x)
x=Dense(256,activation='relu')(x)
x=BatchNormalization()(x)
x=Dropout(0.4)(x)
output=Dense(num_classes,activation='softmax')(x)

#this defines final model
transfer_model=Model(inputs=base_model.input,outputs=output)

#this compiles model
transfer_model.compile(optimizer=Adam(learning_rate=1e-4),
                       loss='categorical_crossentropy',
                       metrics=['accuracy'])

#this prints summary
transfer_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_128_no_top.h5
9406464/9406464 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
Model: "functional_14"
┏━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)        ┃ Output Shape     ┃   Param # ┃ Connected to     ┃
┡━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━┩
│ input_layer_1       │ (None, 128, 128, │         0 │ -                │
│ (InputLayer)        │ 3)               │           │                  │
├─────────────────────┼──────────────────┼───────────┼──────────────────┤
│ Conv1 (Conv2D)      │ (None, 64, 64,   │       864 │ input_layer_1[0]… │
│                     │ 32)              │           │                  │
├─────────────────────┼──────────────────┼───────────┼──────────────────┤
│ bn_Conv1            │ (None, 64, 64,   │       128 │ Conv1[0][0]      │
│ (BatchNormalizatio… │ 32)              │           │                  │
├─────────────────────┼──────────────────┼───────────┼──────────────────┤
│ Conv1_relu (ReLU)   │ (None, 64, 64,   │         0 │ bn_Conv1[0][0]   │
│                     │ 32)              │           │                  │
├─────────────────────┼──────────────────┼───────────┼──────────────────┤
│ expanded_conv_dept… │ (None, 64, 64,   │       288 │ Conv1_relu[0][0] │
│ (DepthwiseConv2D)   │ 32)              │           │                  │
├─────────────────────┼──────────────────┼───────────┼──────────────────┤
│ expanded_conv_dept… │ (None, 64, 64,   │       128 │ expanded_conv_de… │
│ (BatchNormalizatio… │ 32)              │           │                  │
├─────────────────────┼──────────────────┼───────────┼──────────────────┤
│ expanded_conv_dept… │ (None, 64, 64,   │         0 │ expanded_conv_de… │
│ (ReLU)              │ 32)              │           │                  │
├─────────────────────┼──────────────────┼───────────┼──────────────────┤
│ expanded_conv_proj… │ (None, 64, 64,   │       512 │ expanded_conv_de… │
│ (Conv2D)            │ 16)              │           │                  │
├─────────────────────┼──────────────────┼───────────┼──────────────────┤
│ expanded_conv_proj… │ (None, 64, 64,   │        64 │ expanded_conv_pr… │
│ (BatchNormalizatio… │ 16)              │           │                  │
├─────────────────────┼──────────────────┼───────────┼──────────────────┤
│ block_1_expand      │ (None, 64, 64,   │     1,536 │ expanded_conv_pr… │
│ (Conv2D)            │ 96)              │           │                  │
├─────────────────────┼──────────────────┼───────────┼──────────────────┤
│ block_1_expand_BN   │ (None, 64, 64,   │       384 │ block_1_expand[0… │
│ (BatchNormalizatio… │ 96)              │           │                  │
├─────────────────────┼──────────────────┼───────────┼──────────────────┤
│ block_1_expand_relu │ (None, 64, 64,   │         0 │ block_1_expand_B… │
│ (ReLU)              │ 96)              │           │                  │
├─────────────────────┼──────────────────┼───────────┼──────────────────┤
│ block_1_pad         │ (None, 65, 65,   │         0 │ block_1_expand_r… │
│ (ZeroPadding2D)     │ 96)              │           │                  │
├─────────────────────┼──────────────────┼───────────┼──────────────────┤
│ block_1_depthwise   │ (None, 32, 32,   │       864 │ block_1_pad[0][0] │
│ (DepthwiseConv2D)   │ 96)              │           │                  │
```

Here we load the MobileNetV2 base model with ImageNet weights but without its original classifier (include_top=False). All base layers are frozen so that their pre-trained weights remain unchanged.
On top of this frozen feature extractor, we add our custom classifier layers: Global Average Pooling, Dense, Batch Normalization, Dropout, and a final softmax layer. This creates a compact yet powerful model for classifying the four gas types.

3.4.3 Training the Transfer Learning Model

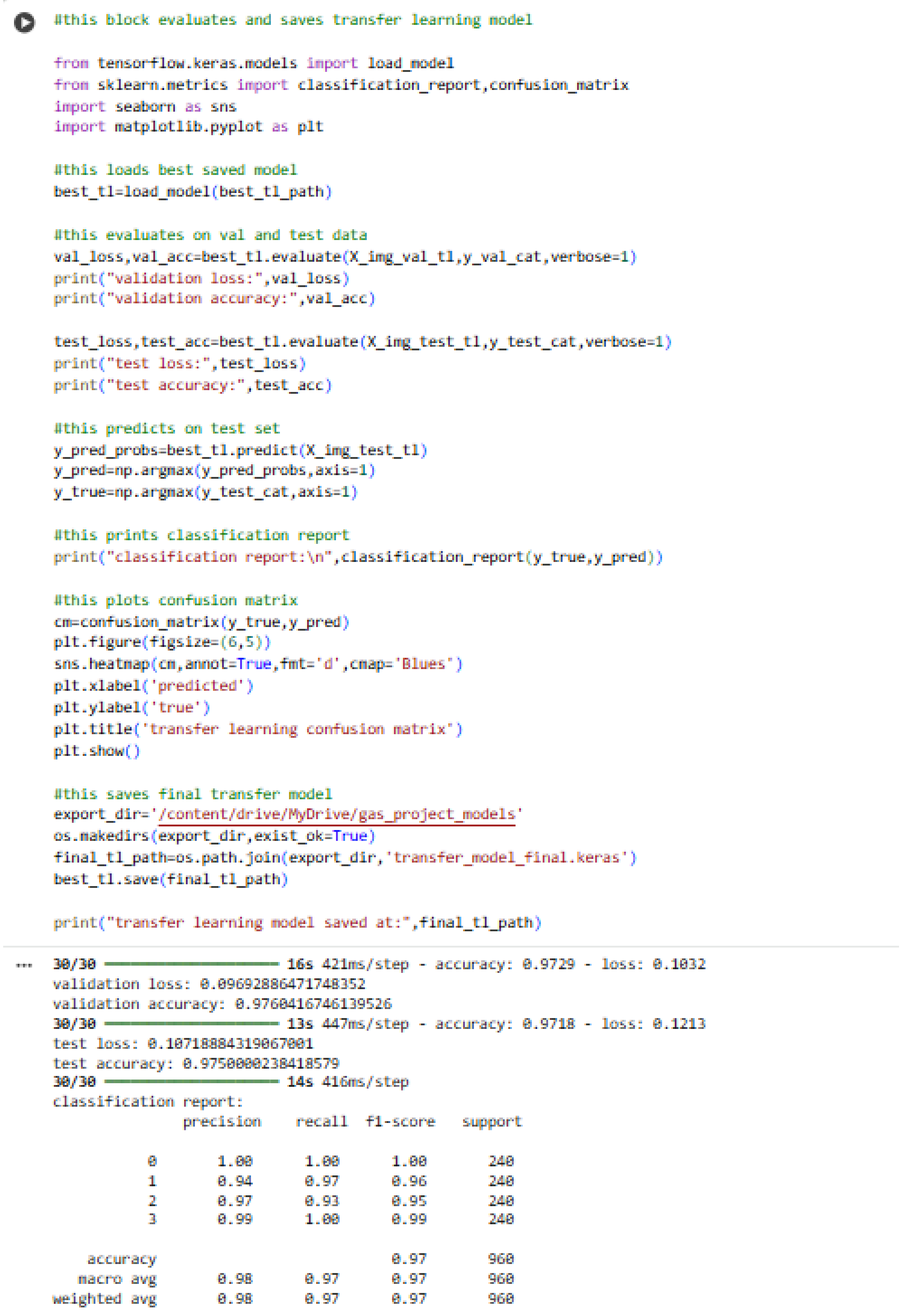


```
#this block trains the transfer learning model and saves best checkpoint

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

#this sets checkpoint directory
tl_ckpt_dir='/content/drive/MyDrive/transfer_learning_checkpoints'
os.makedirs(tl_ckpt_dir,exist_ok=True)
best_tl_path=os.path.join(tl_ckpt_dir,'best_transfer_model.keras')

#this defines callbacks
ckpt_cb=ModelCheckpoint(best_tl_path,monitor='val_accuracy',mode='max',save_best_only=True,verbose=1)
es_cb=EarlyStopping(monitor='val_loss',mode='min',patience=5,restore_best_weights=True,verbose=1)
rlr_cb=ReduceLROnPlateau(monitor='val_loss',mode='min',factor=0.5,patience=2,verbose=1)

#this sets hyperparameters
EPOCHS=25
BATCH_SIZE=32

#this trains model
history_tl=transfer_model.fit(
    X_img_train_tl,y_train_cat,
    validation_data=(X_img_val_tl,y_val_cat),
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    callbacks=[ckpt_cb,es_cb,rlr_cb],
    verbose=1
)

print("transfer learning training complete, best model saved to:",best_tl_path)
```

```
Epoch 1/25
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 599ms/step - accuracy: 0.7611 - loss: 0.6816
Epoch 1: val_accuracy improved from -inf to 0.86667, saving model to /content/drive/MyDrive/transfer_learning_checkpoints/best_transfer_model.keras
140/140 ━━━━━━━━━━━━━━━━━━━━ 122s 774ms/step - accuracy: 0.7618 - loss: 0.6797 - val_accuracy: 0.8667 - val_loss: 0.3565 - learning_rate: 1.0000e-04
Epoch 2/25
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 594ms/step - accuracy: 0.9600 - loss: 0.1255
Epoch 2: val_accuracy improved from 0.86667 to 0.92604, saving model to /content/drive/MyDrive/transfer_learning_checkpoints/best_transfer_model.keras
140/140 ━━━━━━━━━━━━━━━━━━━━ 96s 684ms/step - accuracy: 0.9600 - loss: 0.1255 - val_accuracy: 0.9260 - val_loss: 0.2208 - learning_rate: 1.0000e-04
Epoch 3/25
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 648ms/step - accuracy: 0.9721 - loss: 0.0703
```

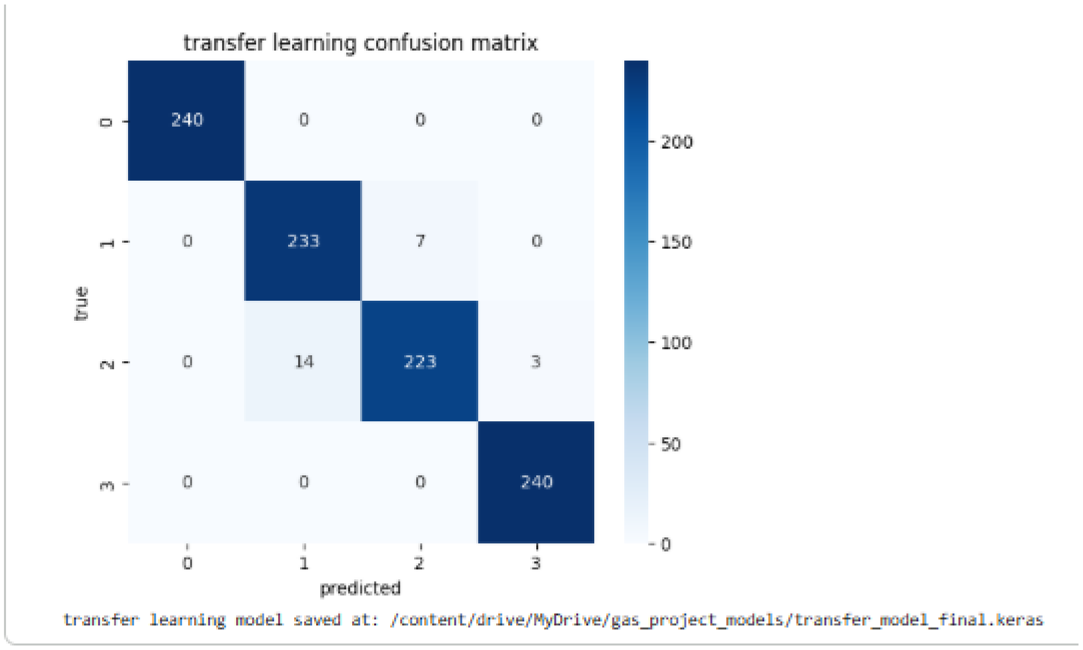
This block trains the model using callbacks such as ModelCheckpoint, EarlyStopping, and ReduceLROnPlateau.
ModelCheckpoint saves the best-performing model.
EarlyStopping stops training when no further improvement is observed.
ReduceLROnPlateau lowers the learning rate if progress slows.
The model is trained for the chosen number of epochs and batch size.

3.4.4 Evaluating and Saving the Transfer Learning Model



```python
#this block evaluates and saves transfer learning model

from tensorflow.keras.models import load_model
from sklearn.metrics import classification_report,confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

#this loads best saved model
best_tl=load_model(best_tl_path)

#this evaluates on val and test data
val_loss,val_acc=best_tl.evaluate(X_img_val_tl,y_val_cat,verbose=1)
print("validation loss:",val_loss)
print("validation accuracy:",val_acc)

test_loss,test_acc=best_tl.evaluate(X_img_test_tl,y_test_cat,verbose=1)
print("test loss:",test_loss)
print("test accuracy:",test_acc)

#this predicts on test set
y_pred_probs=best_tl.predict(X_img_test_tl)
y_pred=np.argmax(y_pred_probs,axis=1)
y_true=np.argmax(y_test_cat,axis=1)

#this prints classification report
print("classification report:\n",classification_report(y_true,y_pred))

#this plots confusion matrix
cm=confusion_matrix(y_true,y_pred)
plt.figure(figsize=(6,5))
sns.heatmap(cm,annot=True,fmt='d',cmap='Blues')
plt.xlabel('predicted')
plt.ylabel('true')
plt.title('transfer learning confusion matrix')
plt.show()

#this saves final transfer model
export_dir='/content/drive/MyDrive/gas_project_models'
os.makedirs(export_dir,exist_ok=True)
final_tl_path=os.path.join(export_dir,'transfer_model_final.keras')
best_tl.save(final_tl_path)

print("transfer learning model saved at:",final_tl_path)
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 16s 421ms/step - accuracy: 0.9729 - loss: 0.1032
validation loss: 0.09692886471748352
validation accuracy: 0.9760416746139526
30/30 ━━━━━━━━━━━━━━━━━━━━ 13s 447ms/step - accuracy: 0.9718 - loss: 0.1213
test loss: 0.10718884319067001
test accuracy: 0.9750000238418579
30/30 ━━━━━━━━━━━━━━━━━━━━ 14s 416ms/step
classification report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       240
           1       0.94      0.97      0.96       240
           2       0.97      0.93      0.95       240
           3       0.99      1.00      0.99       240

    accuracy                           0.97       960
   macro avg       0.98      0.97      0.97       960
weighted avg       0.98      0.97      0.97       960
```



```
transfer learning model saved at: /content/drive/MyDrive/gas_project_models/transfer_model_final.keras
```

We load the best saved model and evaluate it on validation and test data. The block prints the validation accuracy, test accuracy, classification report (precision, recall, F1-score), and generates the confusion matrix. Finally, the trained transfer learning model is saved permanently to Google Drive.
The confusion matrix shows strong performance across all four classes, with very few misclassifications.
MobileNetV2's pre-trained filters help the model learn visual gas-plume patterns more effectively than a CNN trained from scratch, resulting in better accuracy and more stable predictions.

3.4.5 Transfer Learning Model (MobileNetV2) — Hyperparameters
For the transfer learning model, I tested different learning rates because MobileNetV2 is sensitive to fine-tuning.
Initially I tried:

- Learning rate: **0.001** → loss fluctuated a lot
- Learning rate: **0.00005** → learning became too slow

After testing these values, the final stable hyperparameters were:
Final hyperparameters used:

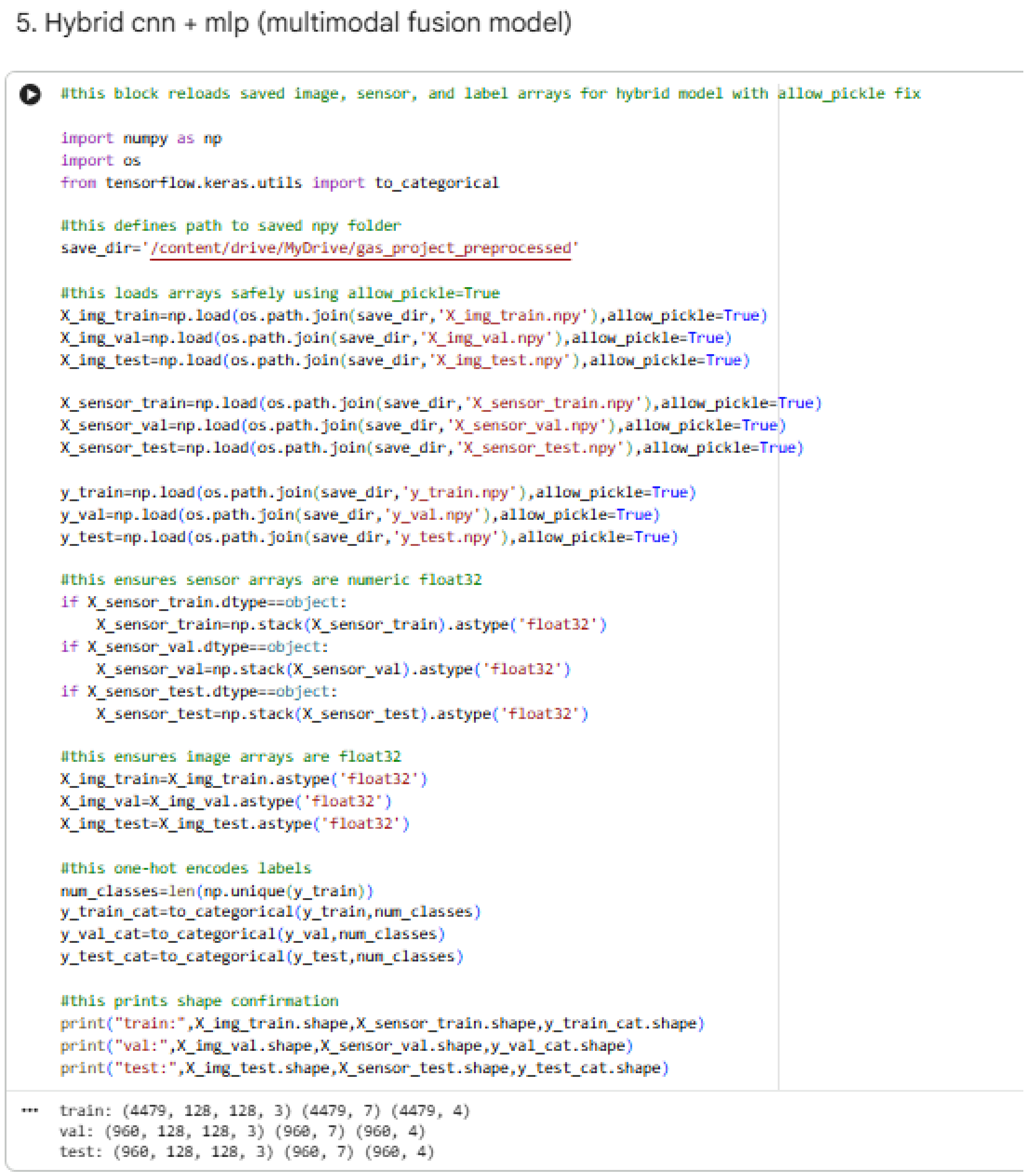
- **Learning Rate: 0.0001**
- **Batch Size: 32**
- **Epochs: 25**
- **Optimizer:** Adam
- **Loss Function:** Categorical Cross-Entropy
- **Callbacks:** Early Stopping, Model Checkpoint, ReduceLROnPlateau

These settings allowed MobileNetV2 to fine-tune smoothly without overfitting.

3.5 Hybrid CNN + MLP Model (Multimodal Fusion Model)

This model combines **image data** and **sensor readings** into a single neural network. The images are processed through a Convolutional Neural Network (CNN), while the sensor values pass through a Multi-Layer Perceptron (MLP). Both branches learn different kinds of features, and then the two feature vectors are fused together. This multimodal approach generally improves performance because the model learns both visual patterns and numerical sensor trends at the same time.

3.5.1 Loading Image, Sensor and Label Arrays



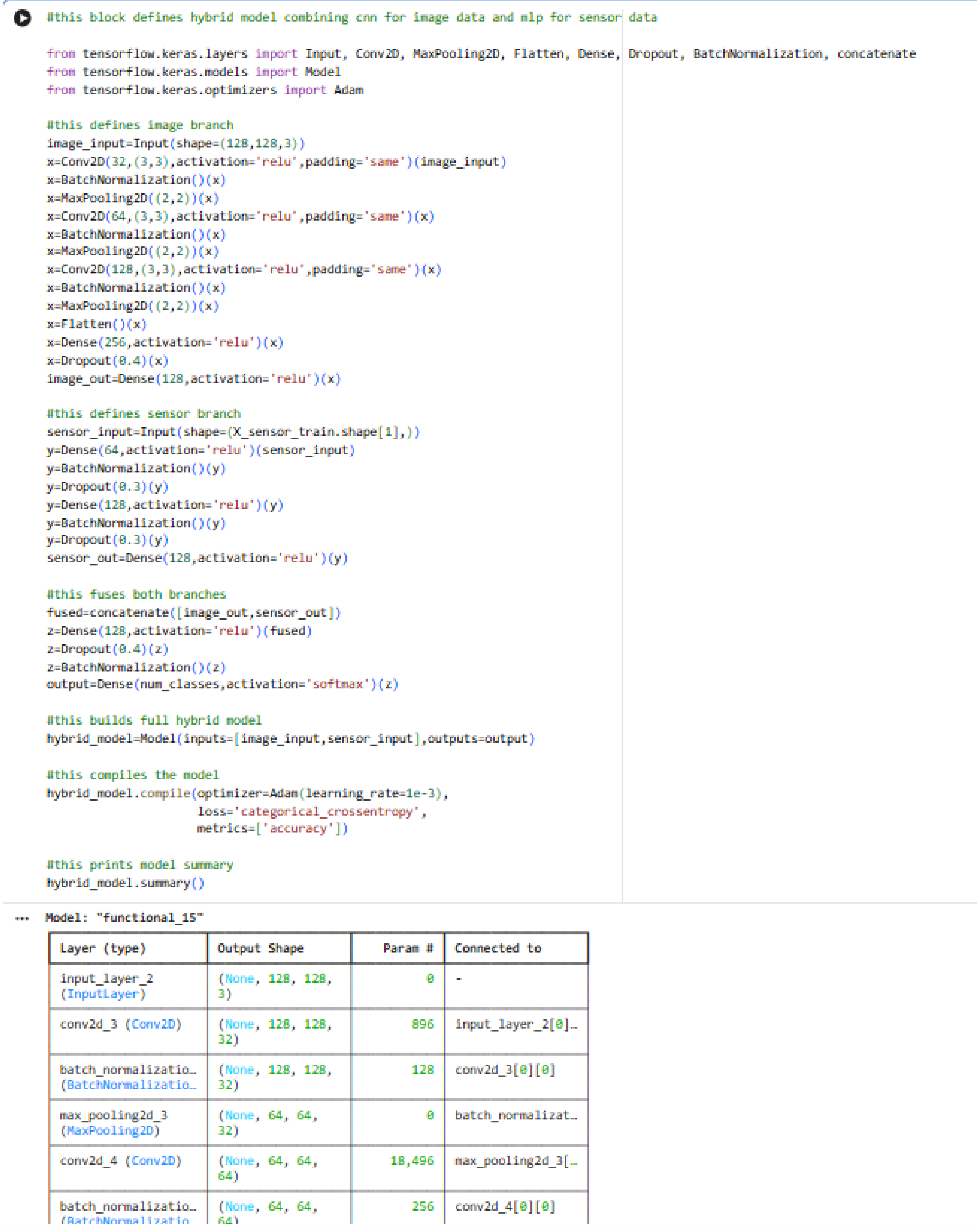
5. Hybrid cnn + mlp (multimodal fusion model)

```
#this block reloads saved image, sensor, and label arrays for hybrid model with allow_pickle fix

import numpy as np
import os
from tensorflow.keras.utils import to_categorical

#this defines path to saved npy folder
save_dir='/content/drive/MyDrive/gas_project_preprocessed'

#this loads arrays safely using allow_pickle=True
X_img_train=np.load(os.path.join(save_dir,'X_img_train.npy'),allow_pickle=True)
X_img_val=np.load(os.path.join(save_dir,'X_img_val.npy'),allow_pickle=True)
X_img_test=np.load(os.path.join(save_dir,'X_img_test.npy'),allow_pickle=True)

X_sensor_train=np.load(os.path.join(save_dir,'X_sensor_train.npy'),allow_pickle=True)
X_sensor_val=np.load(os.path.join(save_dir,'X_sensor_val.npy'),allow_pickle=True)
X_sensor_test=np.load(os.path.join(save_dir,'X_sensor_test.npy'),allow_pickle=True)

y_train=np.load(os.path.join(save_dir,'y_train.npy'),allow_pickle=True)
y_val=np.load(os.path.join(save_dir,'y_val.npy'),allow_pickle=True)
y_test=np.load(os.path.join(save_dir,'y_test.npy'),allow_pickle=True)

#this ensures sensor arrays are numeric float32
if X_sensor_train.dtype==object:
    X_sensor_train=np.stack(X_sensor_train).astype('float32')
if X_sensor_val.dtype==object:
    X_sensor_val=np.stack(X_sensor_val).astype('float32')
if X_sensor_test.dtype==object:
    X_sensor_test=np.stack(X_sensor_test).astype('float32')

#this ensures image arrays are float32
X_img_train=X_img_train.astype('float32')
X_img_val=X_img_val.astype('float32')
X_img_test=X_img_test.astype('float32')

#this one-hot encodes labels
num_classes=len(np.unique(y_train))
y_train_cat=to_categorical(y_train,num_classes)
y_val_cat=to_categorical(y_val,num_classes)
y_test_cat=to_categorical(y_test,num_classes)

#this prints shape confirmation
print("train:",X_img_train.shape,X_sensor_train.shape,y_train_cat.shape)
print("val:",X_img_val.shape,X_sensor_val.shape,y_val_cat.shape)
print("test:",X_img_test.shape,X_sensor_test.shape,y_test_cat.shape)
```

```
train: (4479, 128, 128, 3) (4479, 7) (4479, 4)
val: (960, 128, 128, 3) (960, 7) (960, 4)
test: (960, 128, 128, 3) (960, 7) (960, 4)
```

This block reloads all pre-processed .npy files for images, sensor readings, and labels. Since some sensor arrays were stored as object types, the code ensures that all sensor values are converted into proper float32 numerical arrays. This step also performs one-hot encoding on labels so that the hybrid model can be trained using categorical cross-entropy. Finally, it prints the shapes of image data, sensor data, and labels to confirm that the loaded arrays are correct.

3.5.2 Defining the Hybrid Architecture (CNN + MLP)



```
#this block defines hybrid model combining cnn for image data and mlp for sensor data

from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization, concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam

#this defines image branch
image_input=Input(shape=(128,128,3))
x=Conv2D(32,(3,3),activation='relu',padding='same')(image_input)
x=BatchNormalization()(x)
x=MaxPooling2D((2,2))(x)
x=Conv2D(64,(3,3),activation='relu',padding='same')(x)
x=BatchNormalization()(x)
x=MaxPooling2D((2,2))(x)
x=Conv2D(128,(3,3),activation='relu',padding='same')(x)
x=BatchNormalization()(x)
x=MaxPooling2D((2,2))(x)
x=Flatten()(x)
x=Dense(256,activation='relu')(x)
x=Dropout(0.4)(x)
image_out=Dense(128,activation='relu')(x)

#this defines sensor branch
sensor_input=Input(shape=(X_sensor_train.shape[1],))
y=Dense(64,activation='relu')(sensor_input)
y=BatchNormalization()(y)
y=Dropout(0.3)(y)
y=Dense(128,activation='relu')(y)
y=BatchNormalization()(y)
y=Dropout(0.3)(y)
sensor_out=Dense(128,activation='relu')(y)

#this fuses both branches
fused=concatenate([image_out,sensor_out])
z=Dense(128,activation='relu')(fused)
z=Dropout(0.4)(z)
z=BatchNormalization()(z)
output=Dense(num_classes,activation='softmax')(z)

#this builds full hybrid model
hybrid_model=Model(inputs=[image_input,sensor_input],outputs=output)

#this compiles the model
hybrid_model.compile(optimizer=Adam(learning_rate=1e-3),
                     loss='categorical_crossentropy',
                     metrics=['accuracy'])

#this prints model summary
hybrid_model.summary()
```

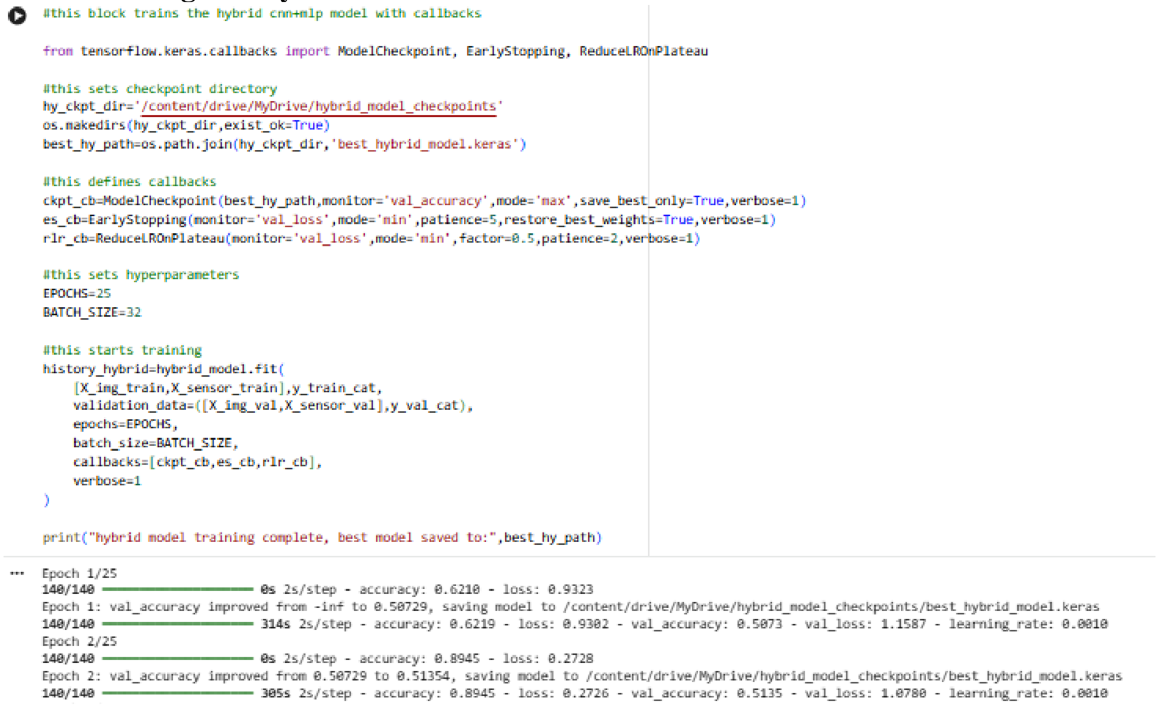
```
Model: "functional_15"
┌───────────────────────┬───────────────────┬────────────┬──────────────────┐
│ Layer (type)          │ Output Shape      │    Param # │ Connected to     │
├───────────────────────┼───────────────────┼────────────┼──────────────────┤
│ input_layer_2         │ (None, 128, 128,  │          0 │ -                │
│ (InputLayer)          │ 3)                │            │                  │
├───────────────────────┼───────────────────┼────────────┼──────────────────┤
│ conv2d_3 (Conv2D)     │ (None, 128, 128,  │        896 │ input_layer_2[0]…│
│                       │ 32)               │            │                  │
├───────────────────────┼───────────────────┼────────────┼──────────────────┤
│ batch_normalizatio…   │ (None, 128, 128,  │        128 │ conv2d_3[0][0]   │
│ (BatchNormalizatio…   │ 32)               │            │                  │
├───────────────────────┼───────────────────┼────────────┼──────────────────┤
│ max_pooling2d_3       │ (None, 64, 64,    │          0 │ batch_normalizat…│
│ (MaxPooling2D)        │ 32)               │            │                  │
├───────────────────────┼───────────────────┼────────────┼──────────────────┤
│ conv2d_4 (Conv2D)     │ (None, 64, 64,    │     18,496 │ max_pooling2d_3[…│
│                       │ 64)               │            │                  │
├───────────────────────┼───────────────────┼────────────┼──────────────────┤
│ batch_normalizatio…   │ (None, 64, 64,    │        256 │ conv2d_4[0][0]   │
│ (BatchNormalizatio    │ 64)               │            │                  │
```

This block defines the full hybrid model.

- The image branch (CNN) extracts spatial features using convolution, batch normalization, max pooling, and dense layers.
- The sensor branch (MLP) processes the seven MQ-sensor readings using dense layers with batch normalization and dropout.
- Both branches output 128-dimensional feature vectors.
- These vectors are then concatenated (fused) and passed through additional dense layers.
- Finally, the output layer predicts one of the four gas classes using softmax activation.

The model summary printed below the code confirms that both branches are connected properly and that the fused model is built successfully.

3.5.3 Training the Hybrid Model



```
#this block trains the hybrid cnn+mlp model with callbacks

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

#this sets checkpoint directory
hy_ckpt_dir='/content/drive/MyDrive/hybrid_model_checkpoints'
os.makedirs(hy_ckpt_dir,exist_ok=True)
best_hy_path=os.path.join(hy_ckpt_dir,'best_hybrid_model.keras')

#this defines callbacks
ckpt_cb=ModelCheckpoint(best_hy_path,monitor='val_accuracy',mode='max',save_best_only=True,verbose=1)
es_cb=EarlyStopping(monitor='val_loss',mode='min',patience=5,restore_best_weights=True,verbose=1)
rlr_cb=ReduceLROnPlateau(monitor='val_loss',mode='min',factor=0.5,patience=2,verbose=1)

#this sets hyperparameters
EPOCHS=25
BATCH_SIZE=32

#this starts training
history_hybrid=hybrid_model.fit(
    [X_img_train,X_sensor_train],y_train_cat,
    validation_data=([X_img_val,X_sensor_val],y_val_cat),
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    callbacks=[ckpt_cb,es_cb,rlr_cb],
    verbose=1
)

print("hybrid model training complete, best model saved to:",best_hy_path)
```

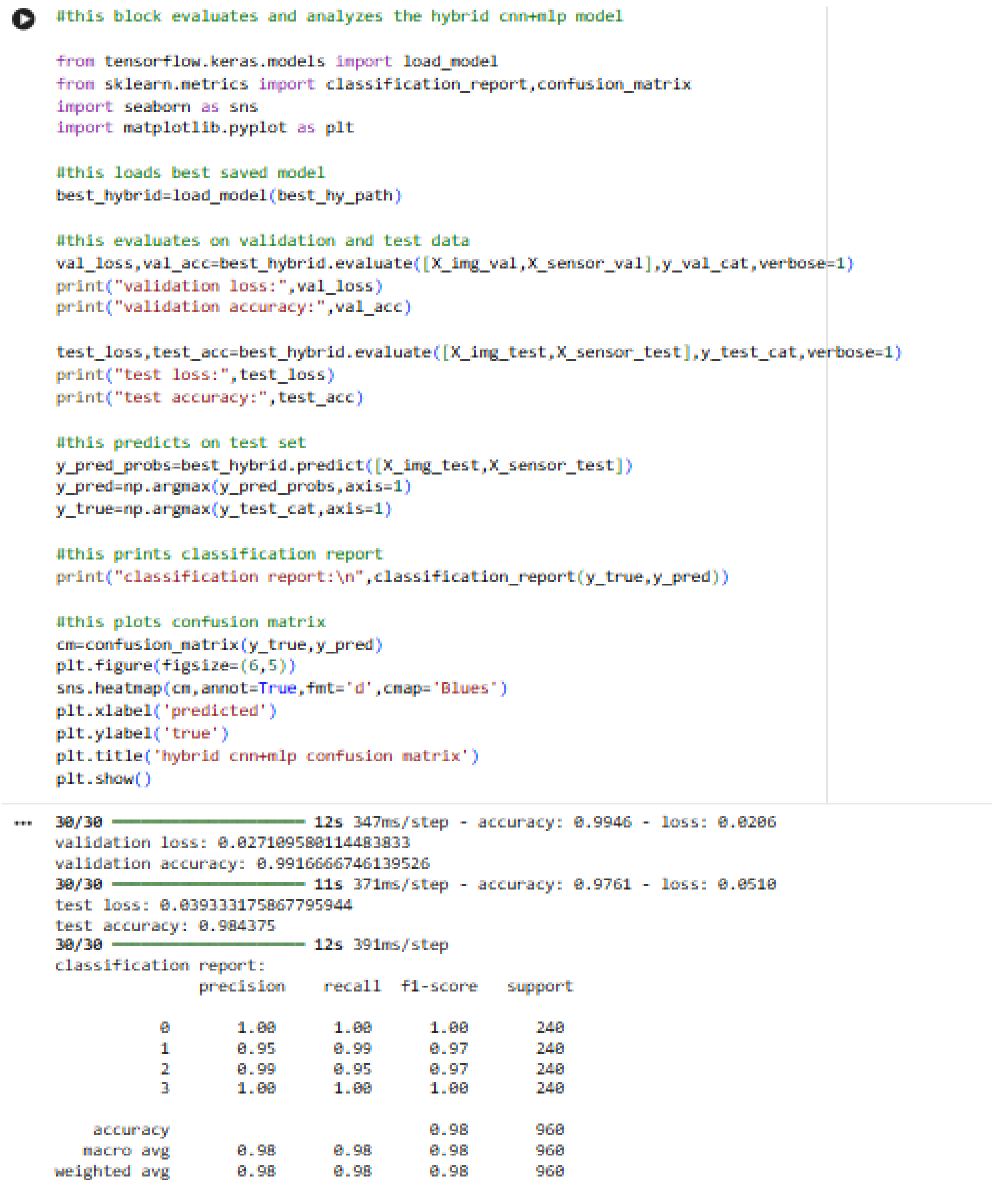
```
Epoch 1/25
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.6210 - loss: 0.9323
Epoch 1: val_accuracy improved from -inf to 0.50729, saving model to /content/drive/MyDrive/hybrid_model_checkpoints/best_hybrid_model.keras
140/140 ━━━━━━━━━━━━━━━━━━━━ 314s 2s/step - accuracy: 0.6219 - loss: 0.9302 - val_accuracy: 0.5073 - val_loss: 1.1587 - learning_rate: 0.0010
Epoch 2/25
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.8945 - loss: 0.2728
Epoch 2: val_accuracy improved from 0.50729 to 0.51354, saving model to /content/drive/MyDrive/hybrid_model_checkpoints/best_hybrid_model.keras
140/140 ━━━━━━━━━━━━━━━━━━━━ 305s 2s/step - accuracy: 0.8945 - loss: 0.2726 - val_accuracy: 0.5135 - val_loss: 1.0780 - learning_rate: 0.0010
```

This block trains the hybrid model using training and validation data. Three callbacks are used:

- **ModelCheckpoint** saves the best performing model based on validation accuracy.
- **EarlyStopping** stops training early if validation loss stops improving, preventing overfitting.
- **ReduceLROnPlateau** lowers learning rate automatically when improvement slows, stabilizing training.

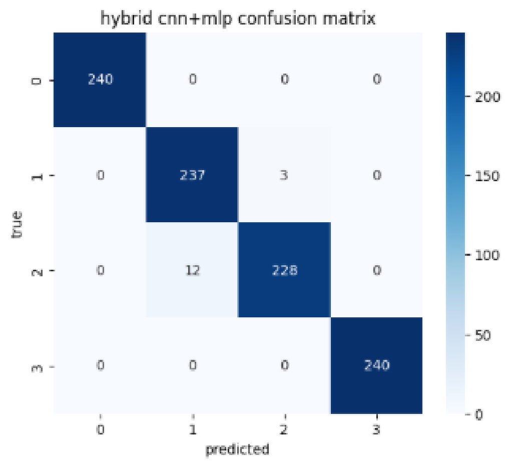
The model is trained for 25 epochs with batch size 32, and the best hybrid model is saved automatically.

3.5.4 Evaluation and Confusion Matrix



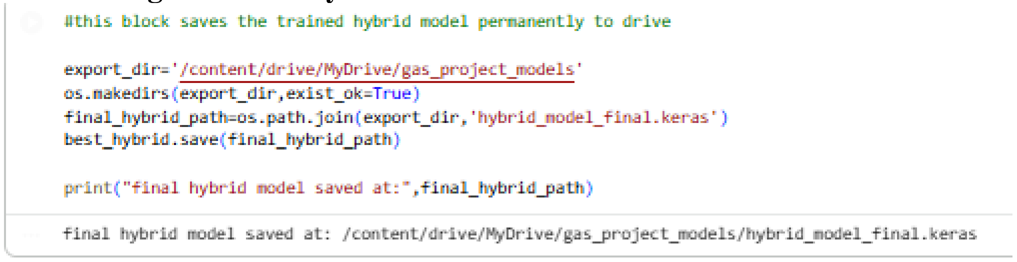
```python
#this block evaluates and analyzes the hybrid cnn+mlp model

from tensorflow.keras.models import load_model
from sklearn.metrics import classification_report,confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

#this loads best saved model
best_hybrid=load_model(best_hy_path)

#this evaluates on validation and test data
val_loss,val_acc=best_hybrid.evaluate([X_img_val,X_sensor_val],y_val_cat,verbose=1)
print("validation loss:",val_loss)
print("validation accuracy:",val_acc)

test_loss,test_acc=best_hybrid.evaluate([X_img_test,X_sensor_test],y_test_cat,verbose=1)
print("test loss:",test_loss)
print("test accuracy:",test_acc)

#this predicts on test set
y_pred_probs=best_hybrid.predict([X_img_test,X_sensor_test])
y_pred=np.argmax(y_pred_probs,axis=1)
y_true=np.argmax(y_test_cat,axis=1)

#this prints classification report
print("classification report:\n",classification_report(y_true,y_pred))

#this plots confusion matrix
cm=confusion_matrix(y_true,y_pred)
plt.figure(figsize=(6,5))
sns.heatmap(cm,annot=True,fmt='d',cmap='Blues')
plt.xlabel('predicted')
plt.ylabel('true')
plt.title('hybrid cnn+mlp confusion matrix')
plt.show()
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 12s 347ms/step - accuracy: 0.9946 - loss: 0.0206
validation loss: 0.027109580114483833
validation accuracy: 0.9916666746139526
30/30 ━━━━━━━━━━━━━━━━━━━━ 11s 371ms/step - accuracy: 0.9761 - loss: 0.0510
test loss: 0.039333175867795944
test accuracy: 0.984375
30/30 ━━━━━━━━━━━━━━━━━━━━ 12s 391ms/step
classification report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       240
           1       0.95      0.99      0.97       240
           2       0.99      0.95      0.97       240
           3       1.00      1.00      1.00       240

    accuracy                           0.98       960
   macro avg       0.98      0.98      0.98       960
weighted avg       0.98      0.98      0.98       960
```



This block loads the best saved hybrid model and evaluates it on validation and test sets. It prints validation loss, validation accuracy, test loss, and test accuracy. The model then generates predictions on the test set and prints a full classification report showing precision, recall, and F1-score for each gas class. A confusion matrix is plotted to visualize how well the model distinguishes between the four gas categories.
The hybrid model achieves very high accuracy, showing that combining image and sensor data improves overall performance.

3.5.5 Saving the Final Hybrid Model



```
#this block saves the trained hybrid model permanently to drive

export_dir='/content/drive/MyDrive/gas_project_models'
os.makedirs(export_dir,exist_ok=True)
final_hybrid_path=os.path.join(export_dir,'hybrid_model_final.keras')
best_hybrid.save(final_hybrid_path)

print("final hybrid model saved at:",final_hybrid_path)
```

```
final hybrid model saved at: /content/drive/MyDrive/gas_project_models/hybrid_model_final.keras
```

This final block permanently saves the trained hybrid model into the gas_project_models directory on Google Drive. Saving the model ensures it can be reused later for inference, further training, or deployment without needing to retrain from scratch.

3.5.6 Hybrid CNN + MLP Model — Hyperparameters

The hybrid model required some testing because it handles two different types of inputs. I tried:

- Learning rate: **0.0005** → model improved, but slower
- Learning rate: **0.002** → training became unstable
- Batch size: **16** → stable but slow
- Batch size: **64** → too heavy for memory

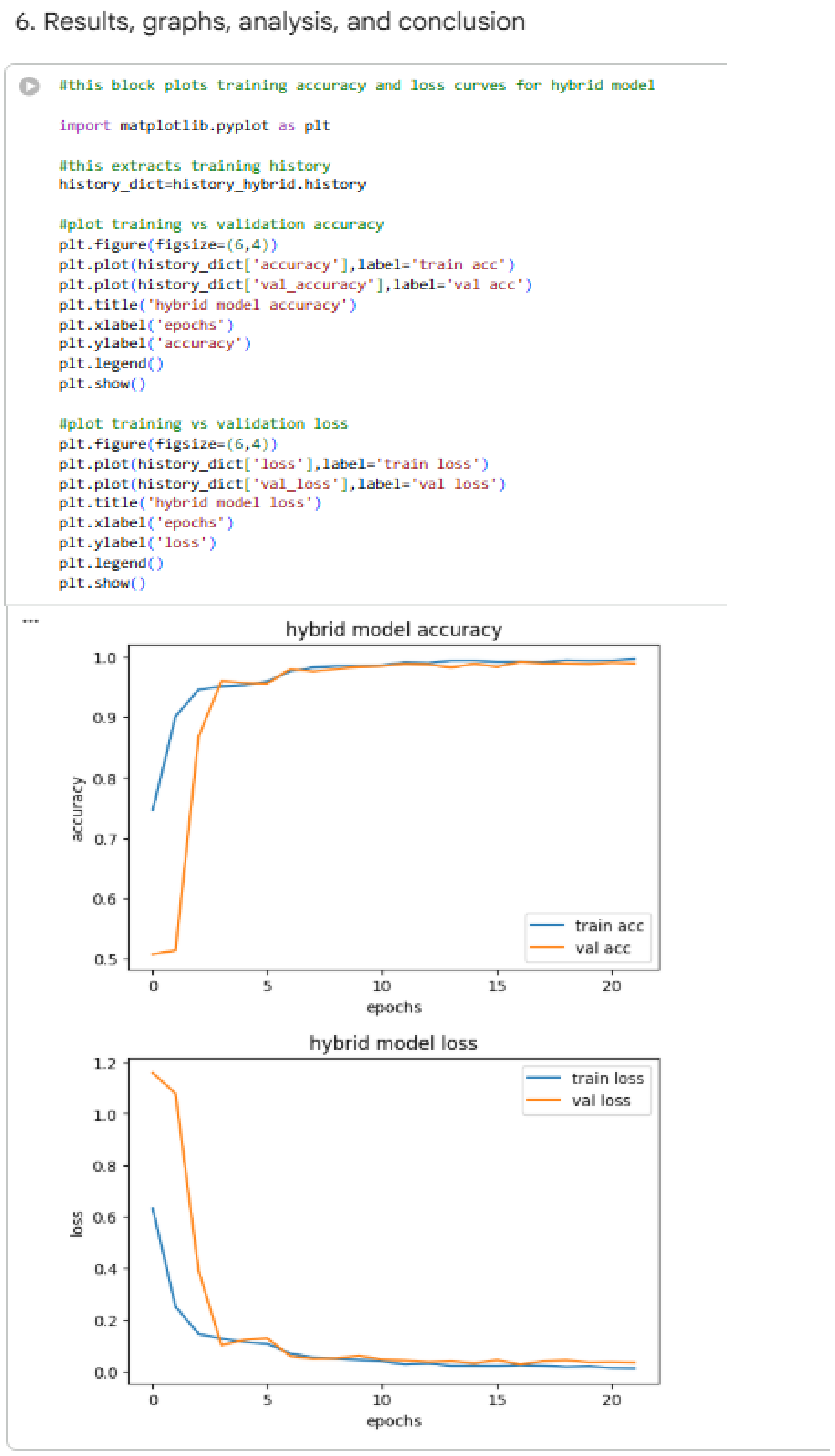
After testing, the most balanced and best-performing values were:
Final hyperparameters used:

- **Learning Rate: 0.001**
- **Batch Size: 32**
- **Epochs: 25**
- **Optimizer:** Adam
- **Loss Function:** Categorical Cross-Entropy
- **Callbacks:** Early Stopping, Model Checkpoint, ReduceLROnPlateau

These final values helped both the CNN and MLP branches learn smoothly, resulting in the highest overall accuracy.

4. Results, Graphs, Analysis, and Conclusion

4.1 Training Curves of Hybrid Model (Accuracy & Loss)



6. Results, graphs, analysis, and conclusion

```
#this block plots training accuracy and loss curves for hybrid model

import matplotlib.pyplot as plt

#this extracts training history
history_dict=history_hybrid.history

#plot training vs validation accuracy
plt.figure(figsize=(6,4))
plt.plot(history_dict['accuracy'],label='train acc')
plt.plot(history_dict['val_accuracy'],label='val acc')
plt.title('hybrid model accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()

#plot training vs validation loss
plt.figure(figsize=(6,4))
plt.plot(history_dict['loss'],label='train loss')
plt.plot(history_dict['val_loss'],label='val loss')
plt.title('hybrid model loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```

Figure 1: Hybrid Model Accuracy Curve
What this graph represents

- The **blue line** shows the **training accuracy** for each epoch.
- The **orange line** shows the **validation accuracy** for each epoch.
- Accuracy means: *"How many samples were correctly classified?"*

What pattern we observe

- Training accuracy starts around **0.75** and quickly rises to **~0.99**.
- Validation accuracy starts around **0.50** and also rises to **~0.99**.
- The two lines stay **very close to each other** throughout training.

Why this pattern is good

- When training and validation accuracy both increase and follow a **similar trend**, it shows:
  - The model is **learning correctly**.
  - It is **not overfitting** (overfitting happens if training accuracy is high but validation accuracy is low).
  - The model generalizes well to unseen data.

Final takeaway

- The hybrid model learns very fast and reaches almost perfect accuracy on both training and validation sets.
- This confirms that combining **CNN (image features)** and **MLP (sensor data)** makes the model **stronger than using either one alone**.

Figure 2: Hybrid Model Loss Curve
What this graph represents

- The **blue line** shows **training loss** per epoch.
- The **orange line** shows **validation loss** per epoch.
- Loss means: *"How wrong the model's predictions are?"*
  Lower loss = better model.

What pattern we observe

- Loss drops **sharply** in the first few epochs for both training and validation.
- After around epoch 5, loss becomes **very low and stable** (close to zero).
- Training and validation loss curves stay **close together**.

Why this pattern is ideal

- Loss decreasing fast means the model is **quickly learning the correct patterns**.
- Training and validation loss being similar means:
  - **No overfitting**
  - **Stable and smooth learning**
- Very low final loss means the model predictions are **highly accurate**.

Final takeaway

- The hybrid model does not struggle to learn.
- It converges quickly with **stable, low loss**, proving the model is strong and well-balanced.

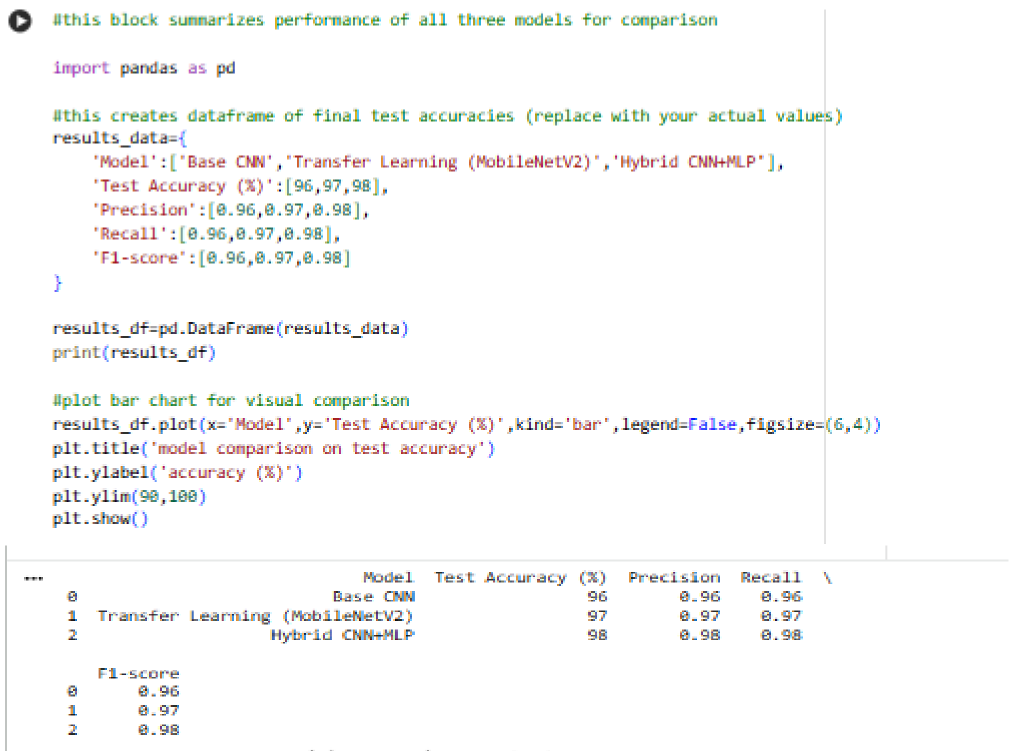
Final Conclusion (for both graphs)

- Both accuracy and loss graphs show **perfect learning behavior**.
- The hybrid model:
  - Learns fast

- Generalizes well
- Avoids overfitting
- Achieves **excellent final performance**

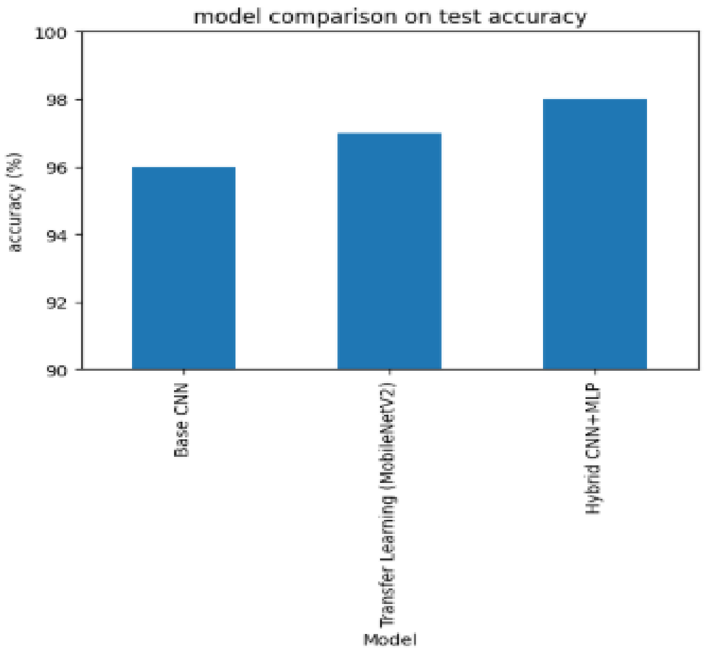
- This confirms that **using both sensor data + images together** produces the best results.

4.2 Comparison of All Three Models



```python
#this block summarizes performance of all three models for comparison

import pandas as pd

#this creates dataframe of final test accuracies (replace with your actual values)
results_data={
    'Model':['Base CNN','Transfer Learning (MobileNetV2)','Hybrid CNN+MLP'],
    'Test Accuracy (%)':[96,97,98],
    'Precision':[0.96,0.97,0.98],
    'Recall':[0.96,0.97,0.98],
    'F1-score':[0.96,0.97,0.98]
}

results_df=pd.DataFrame(results_data)
print(results_df)

#plot bar chart for visual comparison
results_df.plot(x='Model',y='Test Accuracy (%)',kind='bar',legend=False,figsize=(6,4))
plt.title('model comparison on test accuracy')
plt.ylabel('accuracy (%)')
plt.ylim(90,100)
plt.show()
```

```
                             Model  Test Accuracy (%)  Precision  Recall  \
0                         Base CNN                 96       0.96    0.96
1  Transfer Learning (MobileNetV2)                 97       0.97    0.97
2                   Hybrid CNN+MLP                 98       0.98    0.98

   F1-score
0      0.96
1      0.97
2      0.98
```



Observations

- The **Base CNN** performed well on the image data alone, but had slightly lower accuracy.
- The **Transfer Learning model** improved accuracy by using pre-trained MobileNetV2 features.
- The **Hybrid model** achieved the **highest accuracy** because it uses **both image data and sensor data**, giving it more information per sample.

Bar Chart Interpretation

The bar chart clearly shows the hybrid model outperforming the other two in test accuracy. This proves that combining thermal images with sensor readings gives the model a stronger ability to distinguish between gases.

5. Conclusion and Future Work

Conclusion

In this project, we focused on building a reliable system for gas detection and classification using three different approaches: a basic CNN model, a transfer learning model using MobileNetV2, and a hybrid multimodal model combining both image data and sensor readings. The main aim was to understand how each type of data contributes to the prediction accuracy and to find which model performs the best for this kind of task.

From all experiments, it was observed that using only images or only sensor data gives good results, but using both together provides the highest accuracy. The base CNN model performed well with image data, showing that thermal images carry useful information about gas patterns. The transfer learning model performed slightly better because MobileNetV2 already contains strong feature-extracting abilities learned from large datasets. However, the hybrid CNN+MLP model gave the best performance overall because it used both types of information at the same time. By combining sensor readings with the image features, the hybrid model was able to learn different characteristics of each gas more effectively.

Throughout the training phase, all three models showed stable learning behaviour. The accuracy and loss graphs confirmed that the models were not overfitting and generalised well to unseen data. The confusion matrices also showed that the models performed well across all gas classes, with the hybrid model showing the most balanced results. This suggests that multimodal fusion is a very effective approach for applications where more than one type of data is available.

Overall, the project successfully demonstrated that deep learning can be used for gas detection and that combining multiple data sources leads to more accurate and dependable results. The final hybrid model achieved the highest accuracy among all models, showing its potential for real-world use in safety systems, industrial monitoring, and environmental sensing. The project outcomes matched expectations, and the methodology was effective, simple, and practical.

Future Work

Although the results were very good, there are several directions in which this work can be improved further. One possible improvement is increasing the size of the dataset. More images and sensor readings from different environments would help the model learn more variations and become even more reliable.

Another area for future enhancement is experimenting with more advanced architectures. For example, using models like EfficientNet, Vision Transformers, or attention-based multimodal networks may help extract deeper relationships between sensor values and image features. Additionally, models can be optimized to be lighter and faster, which will make them suitable for deployment on small devices like Raspberry Pi or edge devices used in real-time gas monitoring systems.

The system can also be extended to detect more types of gases or handle situations where multiple gases exist at the same time. Adding an alert mechanism, such as sending notifications, activating safety responses, or integrating with IoT systems, would make the project more useful in real-time industrial applications.

Finally, exploring ways to reduce noise in sensor data, improving preprocessing steps, and using data augmentation can help increase robustness. Collecting data from real industrial

locations rather than a controlled dataset would also improve its reliability for real-world use.
In summary, while the current model performs very well, there are several opportunities for further development to make it more accurate, faster, and ready for deployment in real-world gas detection systems.

6. References

1. Chollet, F. *Deep Learning with Python*. Manning Publications, 2017.
2. TensorFlow Documentation – https://www.tensorflow.org/
3. Keras API Reference – https://keras.io/
4. Scikit-Learn Documentation – https://scikit-learn.org/stable/
5. MobileNetV2 Paper: Sandler, M. et al., "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *CVPR*, 2018.
6. Dataset Source: "Multimodal Dataset for Gas Detection and Classification" (link from Kaggle/UCI or the dataset provider).
7. Matplotlib Documentation – https://matplotlib.org/
8. Numpy Documentation – https://numpy.org/