**Tiny CNN Model Training and Deployment Documentation**

**Language:** Python  
**IDE:** PyCharm

**1. Environment Setup**

**1.1 Install PyCharm**

1. Open Google Chrome and navigate to [PyCharm Download](https://www.jetbrains.com/pycharm/download/?section=windows&utm_source=chatgpt.com).
2. Download and install the **Community** or **Professional** edition.
3. Launch PyCharm and create a **New Project**.

**1.2 Configure Python Interpreter**

1. Click on the **Python Interpreter** at the bottom-right corner.
2. Add a Python interpreter (system or virtual environment).
3. Verify the interpreter is successfully added:

(.venv) PS C:\your\_project\_path>

**2. Install Required Libraries**

Run the following commands in the PyCharm terminal or command prompt:

pip install ultralytics  
pip install tensorflow  
pip install matplotlib  
pip install numpy

**3. Project Setup**

1. Create a new Python file in your main project folder:
   * Press Alt+1 → Right-click **Project Folder** → New → Python File
   * Name it tiny\_cnn.py
2. Copy and paste the following code into tiny\_cnn.py.

**4. Tiny CNN Model Implementation**

import tensorflow as tf  
from tensorflow import keras  
from tensorflow.keras import layers  
import pathlib  
import numpy as np  
  
# ========================  
# CONFIG  
# ========================  
IMG\_SIZE = (96, 96) # resize images  
BATCH\_SIZE = 16  
EPOCHS = 10  
  
TRAIN\_DIR = pathlib.Path("C:/Users/bahad/OneDrive/Desktop/ultralytics/bus\_enter\_exist/train")  
VAL\_DIR = pathlib.Path("C:/Users/bahad/OneDrive/Desktop/ultralytics/bus\_enter\_exist/val")  
  
# ========================  
# DATASET  
# ========================  
train\_ds = tf.keras.utils.image\_dataset\_from\_directory(  
 TRAIN\_DIR,  
 labels="inferred",  
 label\_mode="binary",  
 color\_mode="grayscale",  
 image\_size=IMG\_SIZE,  
 batch\_size=BATCH\_SIZE  
)  
  
val\_ds = tf.keras.utils.image\_dataset\_from\_directory(  
 VAL\_DIR,  
 labels="inferred",  
 label\_mode="binary",  
 color\_mode="grayscale",  
 image\_size=IMG\_SIZE,  
 batch\_size=BATCH\_SIZE  
)  
  
# Normalize to [-1,1]  
train\_ds = train\_ds.map(lambda x, y: (x/127.5 - 1.0, y))  
val\_ds = val\_ds.map(lambda x, y: (x/127.5 - 1.0, y))  
  
# Data augmentation  
data\_augmentation = keras.Sequential([  
 layers.RandomFlip("horizontal"),  
 layers.RandomRotation(0.1)  
])  
  
train\_ds = train\_ds.map(lambda x, y: (data\_augmentation(x), y))  
  
# Prefetch for performance  
AUTOTUNE = tf.data.AUTOTUNE  
train\_ds = train\_ds.prefetch(buffer\_size=AUTOTUNE)  
val\_ds = val\_ds.prefetch(buffer\_size=AUTOTUNE)  
  
# ========================  
# MODEL: Tiny CNN  
# ========================  
model = keras.Sequential([  
 layers.Input(shape=(96, 96, 1)),  
 layers.Conv2D(8, (3,3), activation="relu"),  
 layers.MaxPooling2D(),  
 layers.Conv2D(16, (3,3), activation="relu"),  
 layers.MaxPooling2D(),  
 layers.Conv2D(32, (3,3), activation="relu"),  
 layers.Flatten(),  
 layers.Dense(32, activation="relu"),  
 layers.Dropout(0.2),  
 layers.Dense(1, activation="sigmoid") # binary classification  
])  
  
model.compile(  
 optimizer=keras.optimizers.Adam(learning\_rate=1e-4),  
 loss="binary\_crossentropy",  
 metrics=["accuracy"]  
)  
  
model.summary()  
  
# ========================  
# TRAINING  
# ========================  
history = model.fit(  
 train\_ds,  
 validation\_data=val\_ds,  
 epochs=EPOCHS  
)  
  
# ========================  
# SAVE MODEL  
# ========================  
model.save("person\_tinycnn.h5")  
print("✅ Model saved as person\_tinycnn.h5")  
  
# ========================  
# CONVERT TO QUANTIZED TFLITE (INT8)  
# ========================  
# Representative dataset for INT8 quantization  
def representative\_dataset():  
 for images, \_ in train\_ds.take(100):  
 # Must be float32  
 yield [tf.cast(images, tf.float32)]  
  
converter = tf.lite.TFLiteConverter.from\_keras\_model(model)  
converter.optimizations = [tf.lite.Optimize.DEFAULT]  
converter.representative\_dataset = representative\_dataset  
converter.target\_spec.supported\_ops = [tf.lite.OpsSet.TFLITE\_BUILTINS\_INT8]  
converter.inference\_input\_type = tf.int8  
converter.inference\_output\_type = tf.int8  
  
tflite\_model = converter.convert()  
  
with open("person\_tinycnn\_int8.tflite", "wb") as f:  
 f.write(tflite\_model)  
  
print("✅ TinyCNN INT8 quantized model saved as person\_tinycnn\_int8.tflite")

**5. Convert .tflite to .cc for ESP32-CAM Deployment**

tflite\_model\_path = r"C:\Users\bahad\OneDrive\Desktop\ultralytics\person\_tinycnn\_int8.tflite"  
cc\_file\_path = "C:/Users/bahad/OneDrive/Desktop/ultralytics/person\_tinycnn\_int8.cc"  
  
with open(tflite\_model\_path, "rb") as f:  
 data = f.read()  
  
with open(cc\_file\_path, "w") as f:  
 f.write("#include <cstdint>\n\n")  
 f.write(f"const unsigned char waste\_classifier\_int8\_tflite[] = {{\n")  
 for i, byte in enumerate(data):  
 f.write(f"0x{byte:02x},")  
 if (i + 1) % 12 == 0:  
 f.write("\n")  
 f.write("\n};\n")  
 f.write(f"const unsigned int waste\_classifier\_int8\_tflite\_len = {len(data)};\n")

**✅ Notes:**

* Replace file paths with your actual directories.
* .tflite quantized model is optimized for **embedded deployment** on microcontrollers like ESP32-CAM.
* .cc file is ready for inclusion in Arduino or ESP-IDF projects.