

# Edge Intelligence

P Sushma

25MML0050

## Number Plate Detection

Load the number plate dataset

```
import kagglehub
import os
from PIL import Image
import matplotlib.pyplot as plt

path = kagglehub.dataset_download("dataclusterlabs/indian-number-plates-dataset")
print("Path to dataset files:", path)

image_extensions = ('.jpg', '.jpeg', '.png')
all_images = []
for root, dirs, files in os.walk(path):
    for file in files:
        if file.lower().endswith(image_extensions):
            all_images.append(os.path.join(root, file))

if all_images:
    plt.figure(figsize=(12, 8))
    for i in range(min(3, len(all_images))):
        img = Image.open(all_images[i])
        plt.subplot(1, 3, i + 1)
        plt.imshow(img)
        plt.title(f"Sample {i+1}")
        plt.axis('off')
    plt.show()
else:
    print("No images found in the path. Please verify the download.")

...
Warning: Looks like you're using an outdated `kagglehub` version (installed: 0.3.13), please consider upgrading to the latest version (0.4.1).
Downloading from https://www.kaggle.com/api/v1/datasets/download/dataclusterlabs/indian-number-plates-dataset?dataset\_version\_number=3...
100%|██████████| 145M/145M [00:00<00:00, 197MB/s]Extracting files...
Path to dataset files: /root/.cache/kagglehub/datasets/dataclusterlabs/indian-number-plates-dataset/versions/3
```

Sample 1



Sample 3



Sample 2



Resize the image

```
from PIL import Image
import matplotlib.pyplot as plt

if all_images:

    img_path = all_images[0]
    original_img = Image.open(img_path)

    new_size = (200, 200)
    resized_img = original_img.resize(new_size)

    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.imshow(original_img)
    plt.title(f"Original Image ({original_img.width}x{original_img.height})")
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(resized_img)
    plt.title(f"Resized Image ({resized_img.width}x{resized_img.height})")
    plt.axis('off')

    plt.show()
else:
    print("No images available to resize.")

...
Original Image (1968x3071)
```



Resized Image (200x200)



Implement train-test split and print the number of training and testing images.

```
from sklearn.model_selection import train_test_split

if all_images:

    train_images, test_images = train_test_split(all_images, test_size=0.2, random_state=42)

    print(f"Total images: {len(all_images)}")
    print(f"Training images: {len(train_images)}")
    print(f"Testing images: {len(test_images)}")
else:
    print("No images available for train-test split.")

Total images: 47
Training images: 37
Testing images: 10
```

Save the images using pickle

```
| import pickle  
  
# Save train_images  
with open('train_images.pkl', 'wb') as f:  
    pickle.dump(train_images, f)  
print("train_images saved to train_images.pkl")  
  
# Save test_images  
with open('test_images.pkl', 'wb') as f:  
    pickle.dump(test_images, f)  
print("test_images saved to test_images.pkl")  
  
train_images saved to train_images.pkl  
test_images saved to test_images.pkl
```

---

Save the model and weights of the model

```
| import tensorflow as tf  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, Input  
  
IMG_WIDTH = 100  
IMG_HEIGHT = 100  
  
model = Sequential()  
model.add(Input(shape=(IMG_WIDTH, IMG_HEIGHT, 3)))  
model.add(Conv2D(32, (3, 3), activation='relu'))  
model.add(MaxPooling2D((2, 2)))  
model.add(Conv2D(64, (3, 3), activation='relu'))  
model.add(MaxPooling2D((2, 2)))  
model.add(Conv2D(128, (3, 3), activation='relu'))  
model.add(MaxPooling2D((2, 2)))  
model.add(Flatten())  
model.add(Dense(128, activation='relu'))  
model.add(Dropout(0.5))  
model.add(Dense(1, activation='sigmoid'))  
  
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])  
  
print("Model created and compiled successfully.")  
|  
try:  
    model.save_weights('model_weights.weights.h5')  
    print("Initial model weights saved to 'model_weights.weights.h5'")  
except Exception as e:  
    print(f"Error saving initial weights: {e}")  
  
-- Model created and compiled successfully.  
Initial model weights saved to 'model_weights.weights.h5'
```

---

Apply Quantisation and reduce the weight of the model.

```
import numpy as np
import tensorflow as tf
from PIL import Image
import pickle

try:
    with open('train_images.pkl', 'rb') as f:
        train_images = pickle.load(f)
    print(f"Loaded {len(train_images)} training image paths from train_images.pkl")
except FileNotFoundError:
    print("train_images.pkl not found. Please ensure it has been created.")
    train_images = []

def representative_dataset_gen():

    for i in range(len(train_images)):
        if i >= 100:
            break
        img_path = train_images[i]
        try:
            img = Image.open(img_path).resize((IMG_WIDTH, IMG_HEIGHT))
            img = np.array(img, dtype=np.float32)

            if img.ndim == 2:
                img = np.stack([img, img, img], axis=-1)
            elif img.shape[-1] == 4:
                img = img[...,:3]
            img = img / 255.0
            img = np.expand_dims(img, axis=0)
            yield [img]
        except Exception as e:
            print(f"Error processing image {img_path}: {e}")
            continue

converter = tf.lite.TFLiteConverter.from_keras_model(original_model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.representative_dataset = representative_dataset_gen

converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
converter.inference_input_type = tf.int8
converter.inference_output_type = tf.int8
quantized_tflite_model = converter.convert()

print("Model successfully converted to a quantized TFLite model.")

train_images.pkl not found. Please ensure it has been created.
Saved artifact at '/tmp/tmpj73yinwi'. The following endpoints are available:

* Endpoint 'serve'
  args_0 (POSITIONAL_ONLY): TensorSpec(shape=(None, 100, 100, 3), dtype=tf.float32, name='keras_tensor_168')
Output Type:
  TensorSpec(shape=(None, 1), dtype=tf.float32, name=None)
Captures:
137739160565136: TensorSpec(shape=(), dtype=tf.resource, name=None)
137739160566672: TensorSpec(shape=(), dtype=tf.resource, name=None)
137739160563984: TensorSpec(shape=(), dtype=tf.resource, name=None)
137739160563216: TensorSpec(shape=(), dtype=tf.resource, name=None)
137739160565520: TensorSpec(shape=(), dtype=tf.resource, name=None)
137739160567632: TensorSpec(shape=(), dtype=tf.resource, name=None)
137739160563792: TensorSpec(shape=(), dtype=tf.resource, name=None)
137739160567248: TensorSpec(shape=(), dtype=tf.resource, name=None)
137739160567440: TensorSpec(shape=(), dtype=tf.resource, name=None)
137739160566096: TensorSpec(shape=(), dtype=tf.resource, name=None)
/usr/local/lib/python3.12/dist-packages/tensorflow/lite/python/converter.py:854: UserWarning: Statistics for quantized inputs were expected, but not specified; continuing anyway.
warnings.warn('
Model successfully converted to a quantized TFLite model.
```

Print the model weights before and after the quantisation.

```
import os

original_model_path = 'model_weights.weights.h5'
quantized_model_path = 'quantized_model.tflite'

original_size_bytes = os.path.getsize(original_model_path)
quantized_size_bytes = os.path.getsize(quantized_model_path)
|
original_size_kb = original_size_bytes / 1024
quantized_size_kb = quantized_size_bytes / 1024

print(f"Original model weights size: {original_size_kb:.2f} KB")
print(f"Quantized TFLite model size: {quantized_size_kb:.2f} KB")

# Calculate percentage reduction
if original_size_kb > 0:
    reduction_percentage = ((original_size_kb - quantized_size_kb) / original_size_kb) * 100
    print(f"Size reduction through quantization: {reduction_percentage:.2f}%")
else:
    print("Cannot calculate reduction percentage: Original model size is zero.")

Original model weights size: 6797.47 KB
Quantized TFLite model size: 1702.77 KB
Size reduction through quantization: 74.95%
```