```
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt

# ------------------------------------------------------------------------
# Paths and parameters
# ------------------------------------------------------------------------
train_dir = 'train_data/train'     # adjust if using Drive path
img_width, img_height = 48, 48
batch_size = 16
epochs = 15
num_classes = 2

# ------------------------------------------------------------------------
# Data generators
# ------------------------------------------------------------------------
train_datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,
    rotation_range=5,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)

train_gen = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    subset='training'
)

val_gen = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation'
)

# ------------------------------------------------------------------------
# Build model (VGG16 transfer learning)
# ------------------------------------------------------------------------
base = VGG16(weights='imagenet', include_top=False,
             input_shape=(img_width, img_height, 3))
for layer in base.layers[:10]:
    layer.trainable = False

x = Flatten()(base.output)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
output = Dense(num_classes, activation='softmax')(x)

model = Model(inputs=base.input, outputs=output)
model.compile(optimizer=Adam(learning_rate=1e-4),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()

# ------------------------------------------------------------------------
# Train
# ------------------------------------------------------------------------
history = model.fit(
    train_gen,
```

```
    validation_data=val_gen,
    epochs=epochs
)

# ------------------------------------------------------------------
# Plot accuracy
# ------------------------------------------------------------------
plt.figure(figsize=(6,4))
plt.plot(history.history['accuracy'], label='train acc')
plt.plot(history.history['val_accuracy'], label='val acc')
plt.legend()
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.show()

# ------------------------------------------------------------------
# Save model
# ------------------------------------------------------------------
os.makedirs('models', exist_ok=True)
model.save('models/parking_model.h5')
print("Model saved successfully to models/parking_model.h5")
```

```
from google.colab import files
files.download('models/parking_model.h5')
```

```
"""
Optimized detection script for Google Colab.
- Uses GPU (if available)
- Batches predictions for all slots per frame
- Skips frames for faster processing
- Saves annotated output video (parking_detected.mp4)
"""

import cv2
import numpy as np
```

```
import pickle
import os
import tensorflow as tf
from tensorflow.keras.models import load_model

# ------------------------------------------------------------------------
# Configuration
# ------------------------------------------------------------------------

model_path = "models/parking_model.h5"        # trained model uploaded to Colab
video_path = "parking_lot_video.mp4"          # uploaded input video
pkl_path   = "data/slot_positions.pkl"        # uploaded coordinates
output_path = "parking_detected.mp4"

slot_w, slot_h = 120, 45                      # must match annotation
img_w, img_h = 48, 48                         # must match training
frame_interval = 1                            # process every Nth frame (1 = all)

# ------------------------------------------------------------------------
# GPU check
# ------------------------------------------------------------------------
print("GPU devices:", tf.config.list_physical_devices('GPU'))

# ------------------------------------------------------------------------
# Load model and slot positions
# ------------------------------------------------------------------------
print("Loading model and slot coordinates...")
model = load_model(model_path)
with open(pkl_path, "rb") as f:
    slot_positions = pickle.load(f)
print(f"Loaded {len(slot_positions)} slot positions.\n")

# ------------------------------------------------------------------------
# Video setup
# ------------------------------------------------------------------------
cap = cv2.VideoCapture(video_path)
if not cap.isOpened():
    raise IOError("Cannot open video file.")

fps = int(cap.get(cv2.CAP_PROP_FPS))
out_size = (1280, 720)
fourcc = cv2.VideoWriter_fourcc(*"mp4v")
out = cv2.VideoWriter(output_path, fourcc, fps, out_size)

frame_count = 0
processed = 0

print("Processing video...")

while True:
    ret, frame = cap.read()
    if not ret:
        break

    frame_count += 1
    if frame_count % frame_interval != 0:
        continue

    frame = cv2.resize(frame, out_size)
    crops = []
    valid_positions = []

    # Collect all crops for this frame
    for (x, y) in slot_positions:
        roi = frame[y:y + slot_h, x:x + slot_w]
        if roi.size == 0:
            continue
        crop = cv2.resize(roi, (img_w, img_h))
        crops.append(crop)
        valid_positions.append((x, y))
```

```
        if not crops:
            continue

        # Batch predict on GPU
        batch = np.array(crops, dtype="float32") / 255.0
        preds = model.predict(batch, verbose=0)
        labels = np.argmax(preds, axis=1)

        # Draw results
        free_count = int(np.sum(labels == 0))
        occupied_count = int(np.sum(labels == 1))

        for (x, y), label in zip(valid_positions, labels):
            color = (0, 255, 0) if label == 0 else (0, 0, 255)
            cv2.rectangle(frame, (x, y), (x + slot_w, y + slot_h), color, 2)

        # Overlay counts
        cv2.rectangle(frame, (0, 0), (260, 70), (255, 255, 255), -1)
        cv2.putText(frame, f"FREE: {free_count}", (10, 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 200, 0), 2)
        cv2.putText(frame, f"OCCUPIED: {occupied_count}", (10, 60),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)

        out.write(frame)
        processed += 1
        if processed % 10 == 0:
            print(f"Processed {processed} frames...")

    cap.release()
    out.release()
    print(f"Done. Processed {processed} frames. Saved as {output_path}")
```

```
Processed 120 frames...
Processed 130 frames...
Processed 140 frames...
Processed 150 frames...
Processed 160 frames...
Processed 170 frames...
Processed 180 frames...
Processed 190 frames...
Processed 200 frames...
Processed 210 frames...
Processed 220 frames...
Processed 230 frames...
Processed 240 frames...
Processed 250 frames...
Processed 260 frames...
Processed 270 frames...
Processed 280 frames...
Processed 290 frames...
Processed 300 frames...
Processed 310 frames...
Processed 320 frames...
Processed 330 frames...
Processed 340 frames...
Processed 350 frames...
Processed 360 frames...
```