Object Detection Using Tensorflow and MobileNetV2

CPS - 584 | Term Project

Saumya Shah (101737121) | Vincent Durham (101747492)

Introduction:

This report describes the creation and implementation of a simplified object detection system that uses TensorFlow's MobileNetV2 FPNLite architecture. The system is designed to identify bicycles, cats, dogs, men, and women with high accuracy while maintaining efficient real-time performance on mobile devices. The project details the process, challenges, and successful implementation of the system, which enables instant object recognition on handheld devices for practical applications.

Project Description:

Basically, this project focuses on creating a super-fast way to recognize things, and we trained this system to be really good at spotting bicycles, cats, dogs, men, and women. The idea was to make it work on phones so that when you point your phone at something, it quickly tells you what it is, like if it's a bike or a cat! To do this, we worked step by step, teaching the system what these things look like and then making it smart enough to do all this instantly on a phone.

Summary/Steps of the project:

- Collecting images for the dataset
- Labelling all the images
- Downloading the model
- Dividing data into training (80%), testing (10%) and validation data (20%)
- Creating labelmap.pbtxt
- Convert XML dataset file for every image to tfrecord file
- Configuring the model
- Train the model
- Test the model
- Convert the model to tflite file
- Deploy the model using tflite file

Proposed Method:

In this project, we have used a pre-trained model for object detection. The model is MobileNet v2 FPNLite 320, which stands out for object detection due to its efficient and fast architecture, ideal for real-time applications on resource-constrained devices. Its compact design maintains decent accuracy while being lightweight, making it suitable for deployment on devices

with limited computational capabilities. Additionally, its FPNLite architecture enables multi-scale feature fusion, crucial for accurately detecting objects of various sizes within images

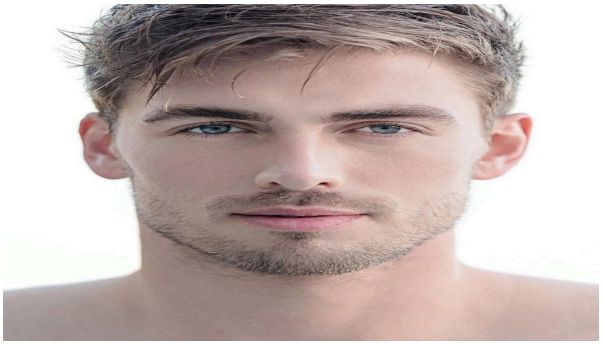
Sources of the Dataset:

For the dataset, we collected over 100 images for each category from Kaggle, and Google. We also took a few photos that contained more than one class. For example, an image containing class Dog, Man and Woman. Sample images for each category is shown below:

Bicycle

Cat

Dog



Man

Woman

Mix photo of Dog and Bicycle

Implementation:

1) Labelling the images:

We used LabelImg for Labelling the images and saved the XML file for every image generated using that. The problem that we encountered here was LabelImg was not working properly in the laptop, so we had to use someone's else system to work on it. Sample shown below:

```
<annotation>
        <folder>Cats</folder>
        <filename>cat (5).jpg</filename>
        <path>/Users/saumya/Desktop/sammy project/Termdata/Cats/cat (5).jpg</path>
        <source>
                <database>Unknown</database>
        </source>
        <size>
                <width>1920</width>
                <height>1080</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>cat</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>371</xmin>
                        <ymin>62</ymin>
                        <xmax>1382</xmax>
                        <ymax>1035</ymax>
                </bndbox>
        </object>
</annotation>
```

2) **Downloading the Model:**

Firstly,, we downloaded all the models from GitHub

```
# Clone the tensorflow models repository from GitHub
!pip uninstall Cython -y # Temporary fix for "No module named 'object_detection'" error
!git clone --depth 1 https://github.com/tensorflow/models
```

```
Found existing installation: Cython 3.0.5
Uninstalling Cython-3.0.5:
  Successfully uninstalled Cython-3.0.5
Cloning into 'models'...
remote: Enumerating objects: 4065, done.
remote: Counting objects: 100% (4065/4065), done.
remote: Compressing objects: 100% (3089/3089), done.
remote: Total 4065 (delta 1186), reused 1944 (delta 916), pack-reused 0
Receiving objects: 100% (4065/4065), 54.71 MiB | 15.95 MiB/s, done.
Resolving deltas: 100% (1186/1186), done.
```

Issue: While working on the project, we got to know that the object detection model is not working with latest tensorflow version, so we had to change the tensorflow version in .ipynb file and the setup file that we downloaded while downloading the pre-trained model.

```
[ ]  # Modify setup.py file to install the tf-models-official repository targeted at TF v2.8.0
     import re
     with open('/content/models/research/object_detection/packages/tf2/setup.py') as f:
         s = f.read()

     with open('/content/models/research/setup.py', 'w') as f:
         # Set fine_tune_checkpoint path
         s = re.sub('tf-models-official>=2.5.1',
                    'tf-models-official==2.8.0', s)
         f.write(s)
```

3) **Dividing data into training (80%), testing data (10%) and validation data(10%):**

To divide the data, we have attached a Python file in the zip folder, name train_val_test_split.py and using that file we can split the data into training, testing and validation dataset.

```
!python train_val_test_split.py
```

```
train_val_test_spli 100%[===================>]   2.74K  --.-KB/s    in 0s

2023-11-28 19:14:27 (29.5 MB/s) - ‘train_val_test_split.py’ saved [2803/2803]

Total images: 508
Images moving to train: 406
Images moving to validation: 50
Images moving to test: 52
```

4) Creating labelmap.pbtxt:

```
### This creates a a "labelmap.txt" file with a list of classes the object detection model will detect.
%%bash
cat <<EOF >> /content/labelmap.txt
bicycle
cat
dog
man
woman
EOF
```

5) Configuring the model:

In this step, we updated the pipeline.config file by changing the number of classes, number of steps, batch size, gave the directory path to training, testing and validation dataset.

```
# Set batch_size
s = re.sub('batch_size: [0-9]+',
           'batch_size: {}'.format(batch_size), s)

# Set training steps, num_steps
s = re.sub('num_steps: [0-9]+',
           'num_steps: {}'.format(num_steps), s)

# Set number of classes num_classes
s = re.sub('num_classes: [0-9]+',
           'num_classes: {}'.format(num_classes), s)

# Change fine-tune checkpoint type from "classification" to "detection"
s = re.sub(
    'fine_tune_checkpoint_type: "classification"', 'fine_tune_checkpoint_type: "{}"'.format('detection'), s)

# If using ssd-mobilenet-v2, reduce learning rate (because it's too high in the default config file)
if chosen_model == 'ssd-mobilenet-v2':
  s = re.sub('learning_rate_base: .8',
             'learning_rate_base: .08', s)

  s = re.sub('warmup_learning_rate: 0.13333',
             'warmup_learning_rate: .026666', s)
```

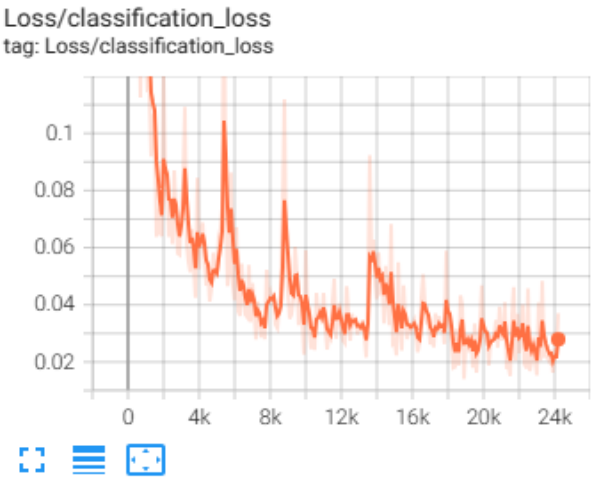
6) Training the model:

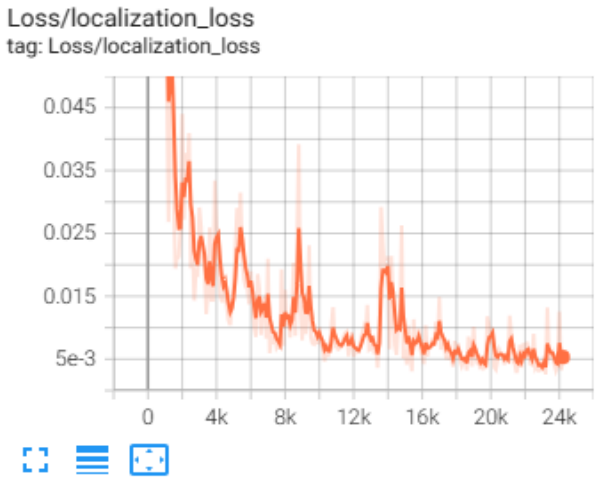
An issue we encounter: Initially, we tried training the model for 10000 steps, however, the accuracy of model was not really good and it was around 50%. Then we tried training the model for 25000 steps and we got an accuracy of 70% and so we initialized num_steps = 25000.

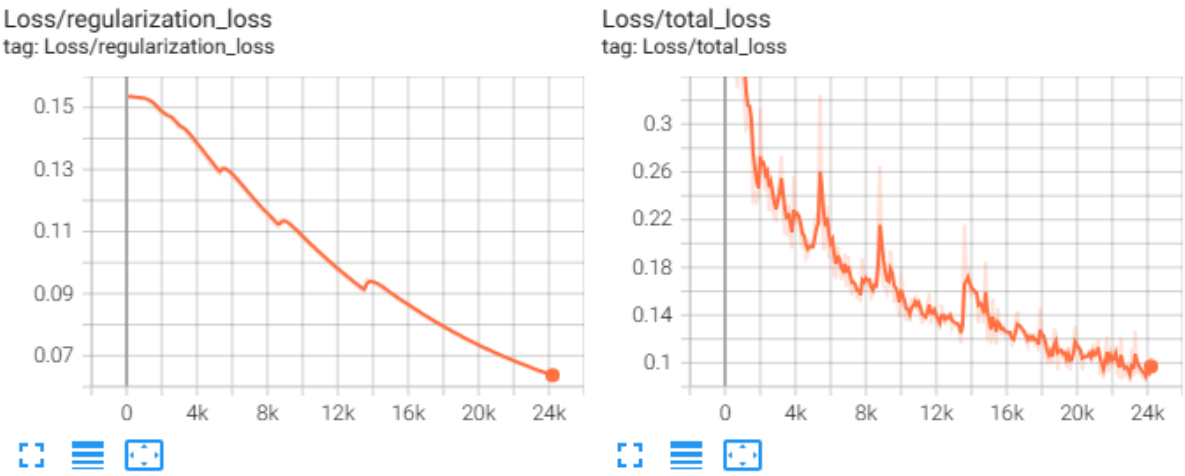
```
# Run training!
!python /content/models/research/object_detection/model_main_tf2.py \
    --pipeline_config_path={pipeline_file} \
    --model_dir={model_dir} \
    --alsologtostderr \
    --num_train_steps={num_steps} \
    --sample_1_of_n_eval_examples=1
```

```
I1128 22:33:33.684747 138856029999744 model_lib_v2.py:708] {'Loss/classification_loss': 0.02347166,
 'Loss/localization_loss': 0.0028754955,
 'Loss/regularization_loss': 0.06230531,
 'Loss/total_loss': 0.08865246,
 'learning_rate': 0.04179459}
INFO:tensorflow:Step 24900 per-step time 0.458s
I1128 22:34:19.507293 138856029999744 model_lib_v2.py:705] Step 24900 per-step time 0.458s
INFO:tensorflow:{'Loss/classification_loss': 0.020297376,
 'Loss/localization_loss': 0.004917179,
 'Loss/regularization_loss': 0.062124528,
 'Loss/total_loss': 0.08733908,
 'learning_rate': 0.04153835}
I1128 22:34:19.507678 138856029999744 model_lib_v2.py:708] {'Loss/classification_loss': 0.020297376,
 'Loss/localization_loss': 0.004917179,
 'Loss/regularization_loss': 0.062124528,
 'Loss/total_loss': 0.08733908,
 'learning_rate': 0.04153835}
INFO:tensorflow:Step 25000 per-step time 0.452s
I1128 22:35:04.743305 138856029999744 model_lib_v2.py:705] Step 25000 per-step time 0.452s
INFO:tensorflow:{'Loss/classification_loss': 0.025273092,
 'Loss/localization_loss': 0.0030157317,
 'Loss/regularization_loss': 0.06194515,
 'Loss/total_loss': 0.090233974,
 'learning_rate': 0.04128206}
I1128 22:35:04.743754 138856029999744 model_lib_v2.py:708] {'Loss/classification_loss': 0.025273092,
 'Loss/localization_loss': 0.0030157317,
 'Loss/regularization_loss': 0.06194515,
 'Loss/total_loss': 0.090233974,
 'learning_rate': 0.04128206}
```

7) Results - Plotting graphs:



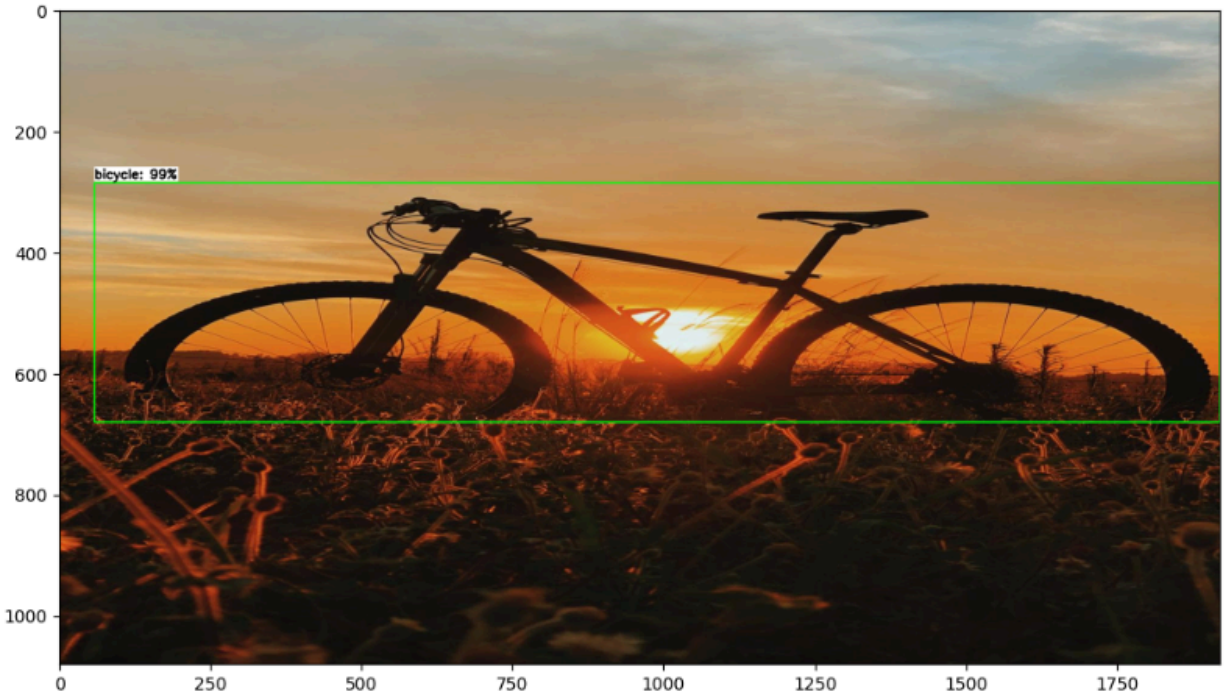


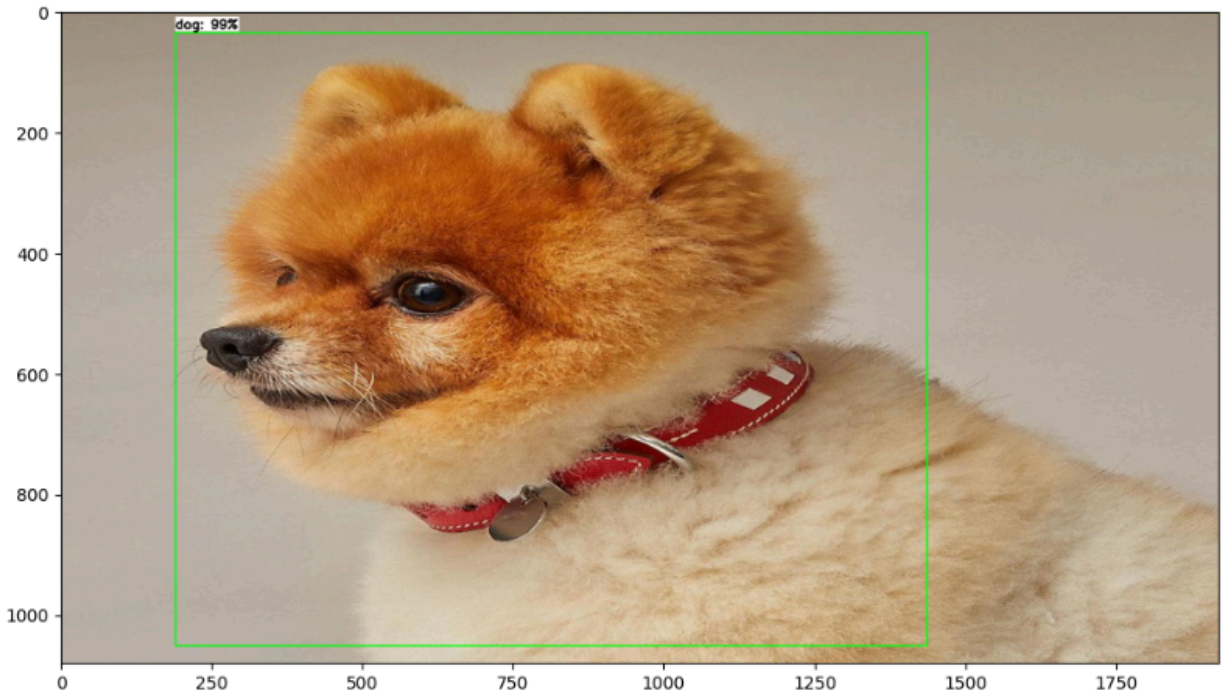


8) Calculating accuracy for different thresholds:

```
Calculating mAP at 0.50 IoU threshold...
99.58% = bicycle AP
63.25% = cat AP
62.86% = dog AP
100.00% = man AP
100.00% = woman AP
mAP = 85.14%
Calculating mAP at 0.55 IoU threshold...
99.58% = bicycle AP
56.86% = cat AP
62.86% = dog AP
100.00% = man AP
100.00% = woman AP
mAP = 83.86%
Calculating mAP at 0.60 IoU threshold...
99.58% = bicycle AP
56.86% = cat AP
62.86% = dog AP
100.00% = man AP
100.00% = woman AP
mAP = 83.86%
Calculating mAP at 0.65 IoU threshold...
99.58% = bicycle AP
48.14% = cat AP
62.86% = dog AP
100.00% = man AP
100.00% = woman AP
mAP = 82.12%
```

9) Testing the model:





10) Converting the model to tflite file using TensorFlowLite:

```
[ ]  # Make a directory to store the trained TFLite model
     !mkdir /content/custom_model_lite
     output_directory = '/content/custom_model_lite'

     # Path to training directory (the conversion script automatically chooses the highest checkpoint file)
     last_model_path = '/content/training'

     !python /content/models/research/object_detection/export_tflite_graph_tf2.py \
         --trained_checkpoint_dir {last_model_path} \
         --output_directory {output_directory} \
         --pipeline_config_path {pipeline_file}
```

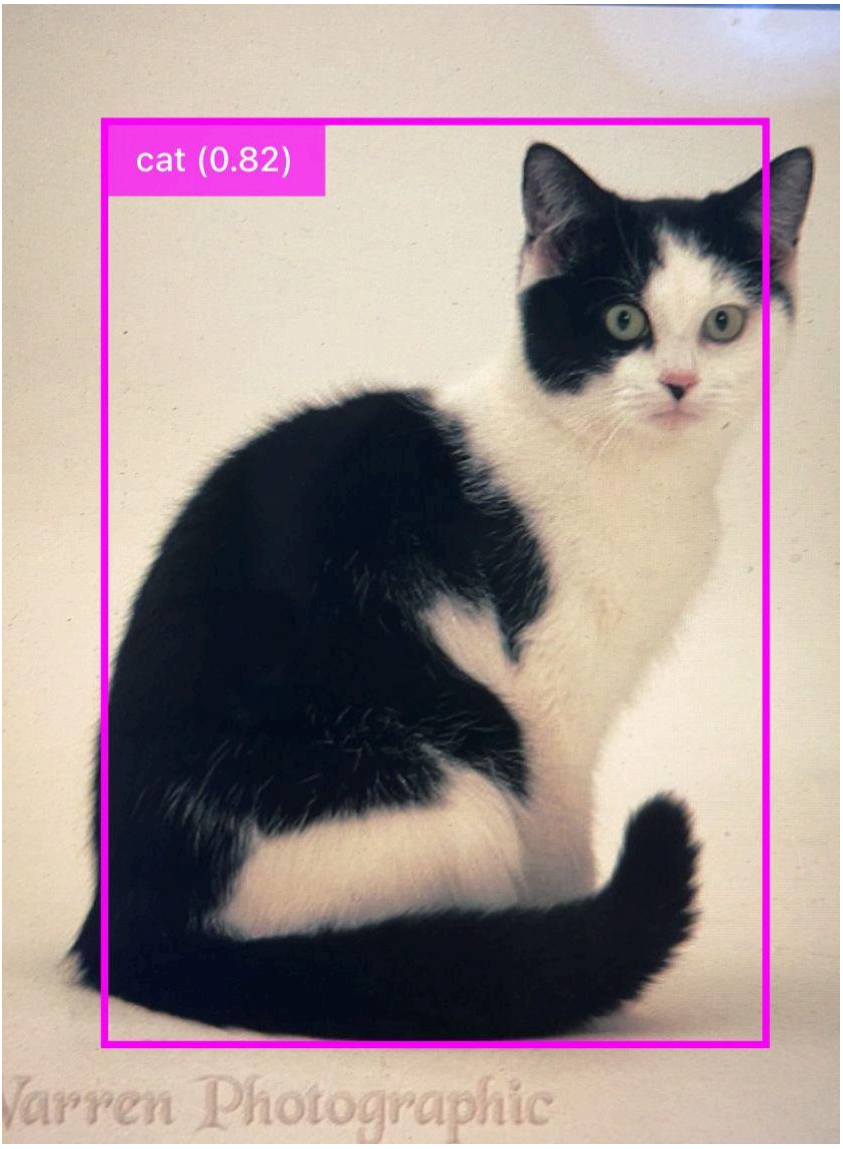
```
     # Move labelmap and pipeline config files into TFLite model folder and zip it up
     !cp /content/labelmap.txt /content/custom_model_lite
     !cp /content/labelmap.pbtxt /content/custom_model_lite
     !cp /content/models/mymodel/pipeline_file.config /content/custom_model_lite

     %cd /content
     !zip -r custom_model_lite.zip custom_model_lite
```

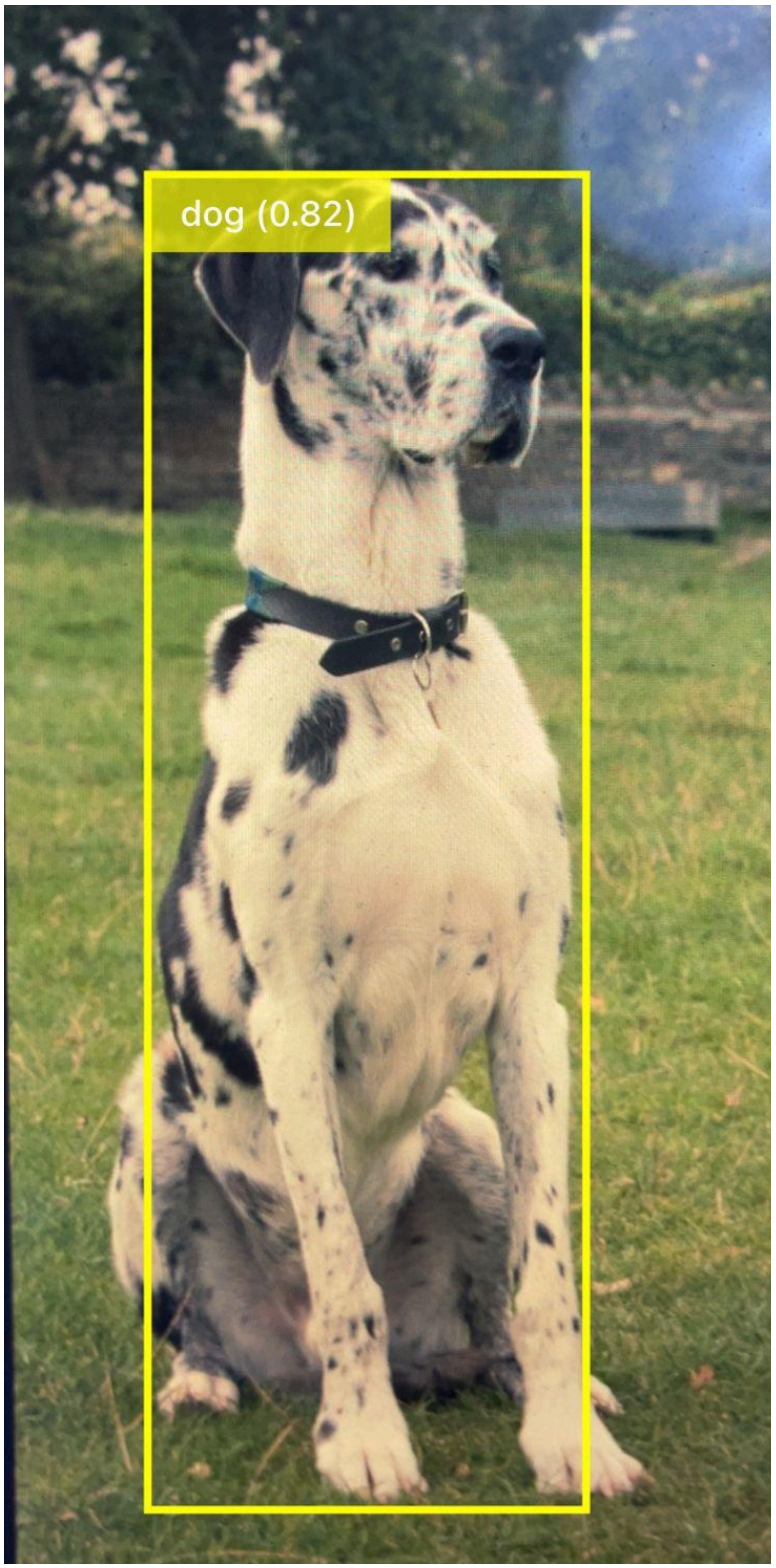
```
/content
  adding: custom_model_lite/ (stored 0%)
  adding: custom_model_lite/labelmap.txt (deflated 4%)
  adding: custom_model_lite/pipeline_file.config (deflated 65%)
  adding: custom_model_lite/detect.tflite (deflated 9%)
  adding: custom_model_lite/labelmap.pbtxt (deflated 58%)
  adding: custom_model_lite/saved_model/ (stored 0%)
  adding: custom_model_lite/saved_model/saved_model.pb (deflated 91%)
  adding: custom_model_lite/saved_model/variables/ (stored 0%)
  adding: custom_model_lite/saved_model/variables/variables.index (deflated 78%)
  adding: custom_model_lite/saved_model/variables/variables.data-00000-of-00001 (deflated 9%)
  adding: custom_model_lite/saved_model/assets/ (stored 0%)
```

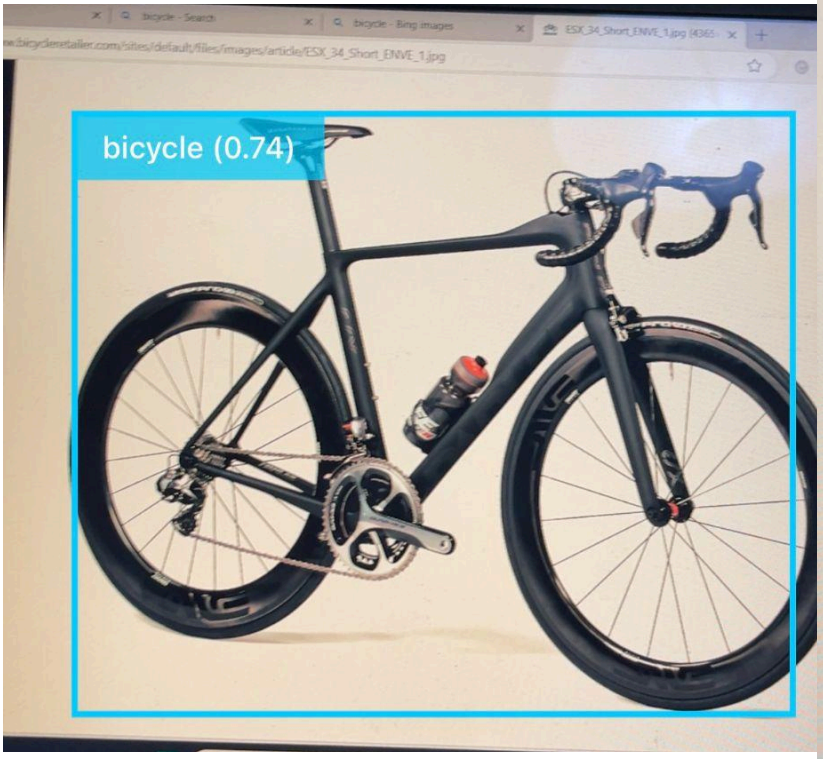
```
[ ]  from google.colab import files

     files.download('/content/custom_model_lite.zip')
```

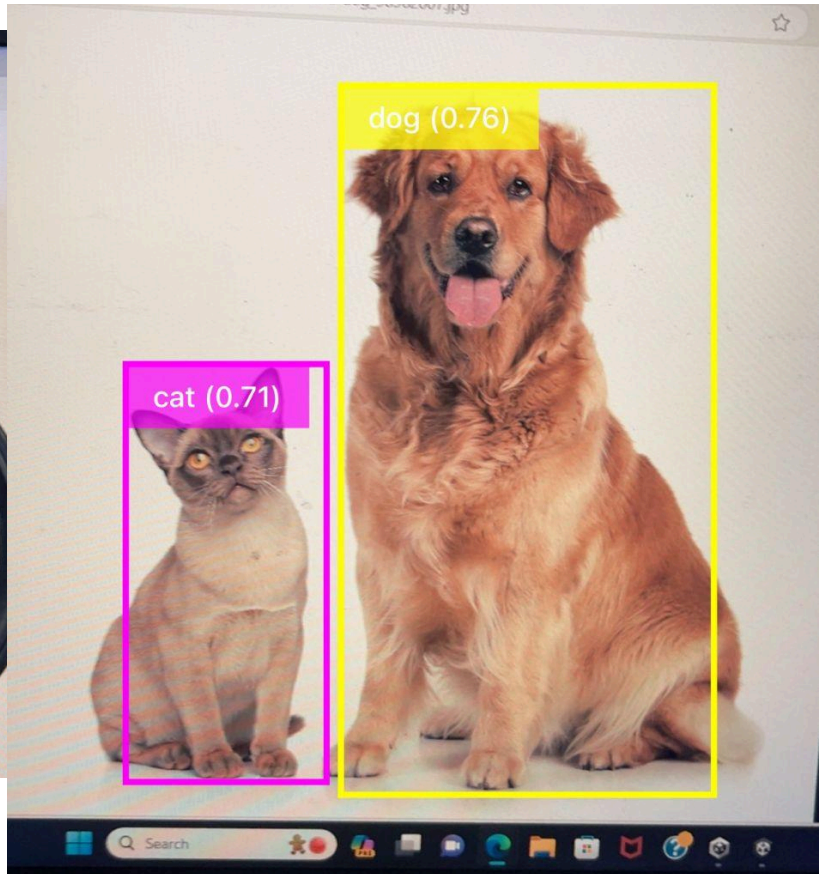
11) Deploying the model using iOS:

To import to an iPhone, everything needed to be moved to a MacOS for testing. We used a Github Object Detection IOS Example to insert our own model into. (examples/lite/examples/object_detection/ios at master · tensorflow/examples · GitHub)


cat (0.82)
Warren Photographic


dog (0.82)





12) Creating the Metadata for the images

To use our model with live images coming in, we had to create metadata from our detect.tflite. The usual way to create metadata isn't currently supported by Python 3.10, so this was somewhat of a headache to do.

```
from google.colab import files
from tflite_support.metadata_writers import object_detector
from tflite_support.metadata_writers import writer_utils

# Upload the TFLite model and labelmap files using the file upload widget
uploaded = files.upload()

# Get the file paths in Colab environment
path_tf = "/content/" + next(iter(uploaded))
path_lb = "/content/" + next(iter(uploaded))
_SAVE_TO_PATH = "/content/metadata.tflite"
_INPUT_NORM_MEAN = 127.5
_INPUT_NORM_STD = 127.5

# Load the TFLite model as bytes.
model_content = writer_utils.load_file(path_tf)

# Create the metadata writer.
writer = object_detector.MetadataWriter.create_for_inference(
    model_content, [_INPUT_NORM_MEAN], [_INPUT_NORM_STD], [path_lb])

# Verify the metadata generated by metadata writer.
print(writer.get_metadata_json())

# Populate the metadata into the model.
writer_utils.save_file(writer.populate(), _SAVE_TO_PATH)
```

Choose Files 2 files

- **detect.tflite**(n/a) - 11509376 bytes, last modified: 12/1/2023 - 100% done
- **labelmap.txt**(text/plain) - 26 bytes, last modified: 12/1/2023 - 100% done

```
Saving detect.tflite to detect.tflite
Saving labelmap.txt to labelmap.txt
{
  "name": "ObjectDetector",
  "description": "Identify which of a known set of objects might be present and provide information about their positions within the giv
  "subgraph_metadata": [
    {
      "input_tensor_metadata": [
        {
          "name": "image",
          "description": "Input image to be detected.",
          "content": {
            "content_properties_type": "ImageProperties",
            "content_properties": {
              "color_space": "RGB"
            }
          },
          "process_units": [
            {
```

13) Difficulties with deploying

Deploying on iOS required an Apple developers license and a good amount of knowledge in the Swift coding language. The metadata challenge was mentioned but another problem was preprocessing and postprocessing the input video data to fit our model's requirements.

Conclusion:

In this project, we seamlessly integrated TensorFlow with MobileNetV2 FPNLite 320 to develop a sophisticated object detection application. By meticulously labeling our dataset with LabelImg and harnessing TensorFlow Lite for model optimization, we achieved a streamlined and efficient deployment. Our final application demonstrated not only the precision of our model in real-time object detection but also the powerful synergy of advanced machine learning tools in transforming theoretical concepts into practical, impactful solutions.