```python
import os
import glob
import numpy as np
import tensorflow as tf

tf.get_logger().setLevel('ERROR')
```

✓ 3.4s Python

Load Dataset

```python
BATCH_SIZE = 4
IMG_SIZE = (256, 256)

image_filenames = glob.glob("D:\Grox's Vault\Laboratory\Repos\Bisa.ai_capstone\Dataset\TRANCOS_v3\images\*.jpg")
label_filenames = glob.glob("D:\Grox's Vault\Laboratory\Repos\Bisa.ai_capstone\Dataset\TRANCOS_v3\cars_count_label\*.txt")
image_filenames = np.array(image_filenames)
label = np.array([])
for filename in label_filenames :
    with open(filename,"r") as file_handler :
        data = file_handler.read()
        label = np.append(label, data)
```

✓ 0.2s Python

```python
image_count = len(image_filenames)

filenames = tf.constant(image_filenames)
labels = tf.constant(label)
dataset = tf.data.Dataset.from_tensor_slices((filenames, labels))

def _parse_function(filename, label):
  image_string = tf.io.read_file(filename)
  image_decoded = tf.image.decode_jpeg(image_string)
  image_resized = tf.image.resize (image_decoded, IMG_SIZE)
  return filename, image_resized, label

list_ds = tf.data.Dataset.list_files("D:\Grox's Vault\Laboratory\Repos\Bisa.ai_capstone\Dataset\TRANCOS_v3\images\*.jpg", shuffle=False)
list_ds = list_ds.shuffle(image_count, reshuffle_each_iteration=False)

class_names = np.array(sorted([item for item in label]))

val_size = int(image_count * 0.2)
train_ds = list_ds.skip(val_size)
```

```python
train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

✓ 0.2s Python

```
996
248
```

```python
def get_label(file_path):
  # Convert the path to a list of path components
  parts = tf.strings.split(file_path, os.path.sep)
  # The second to last is the class-directory
  one_hot = parts[-2] == class_names
  # Integer encode the label
  return tf.argmax(one_hot)

def decode_img(img):
  # Convert the compressed string to a 3D uint8 tensor
  img = tf.io.decode_jpeg(img, channels=3)
  # Resize the image to the desired size
  return tf.image.resize(img, [256, 256])

def process_path(file_path):
  label = get_label(file_path)
  # Load the raw data from the file as a string
  img = tf.io.read_file(file_path)
  img = decode_img(img)
  return img, label
```

✓ 0.3s Python

```python
# Set `num_parallel_calls` so multiple images are loaded/processed in parallel.
train_ds = train_ds.map(process_path, num_parallel_calls=AUTOTUNE)
val_ds = val_ds.map(process_path, num_parallel_calls=AUTOTUNE)
```

✓ 0.2s Python

```python
def configure_for_performance(ds):
  ds = ds.cache()
  ds = ds.shuffle(buffer_size=1000)
  ds = ds.batch(BATCH_SIZE)
  ds = ds.prefetch(buffer_size=AUTOTUNE)
  return ds

train_ds = configure_for_performance(train_ds)
val_ds = configure_for_performance(val_ds)
```

```python
IMG_SHAPE = IMG_SIZE + (3,)
preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input
base_model = tf.keras.applications.MobileNetV2(input_shape=(IMG_SHAPE),
                                               include_top=False,
                                               weights='imagenet')
prediction_layer = tf.keras.layers.Dense(1, activation='linear')
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()


inputs = tf.keras.Input(shape=(256, 256, 3))
x = preprocess_input(inputs)
x = base_model(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)

model.compile(
        optimizer='adam',
        loss=tf.keras.losses.categorical_crossentropy,
        metrics=['accuracy'])
model.summary()
```

✓ 0.9s Python

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 256, 256, 3)]     0

 tf.math.truediv (TFOpLambda  (None, 256, 256, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 256, 256, 3)      0
 a)

 mobilenetv2_1.00_224 (Funct  (None, 8, 8, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1280)              0
```

```
model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=3
)
```

[9] ✓ 2m 43.7s Python

```
Epoch 1/3
249/249 [==============================] - 63s 215ms/step - loss: 0.0000e+00 - accuracy: 0.9769 - val_loss: 0.0000e+00 - val_accuracy: 0.9960
Epoch 2/3
 24/249 [=>............................] - ETA: 41s - loss: 0.0000e+00 - accuracy: 0.9896

<keras.callbacks.History at 0x13d73281d30>
```