PYTHON PROJECT

ON

COMPUTER VISION APPLICATION

By

Neraj OblaKumarBabu and Manisha Pulluru.

PET DETECTOR USING TENSORFLOW MODEL

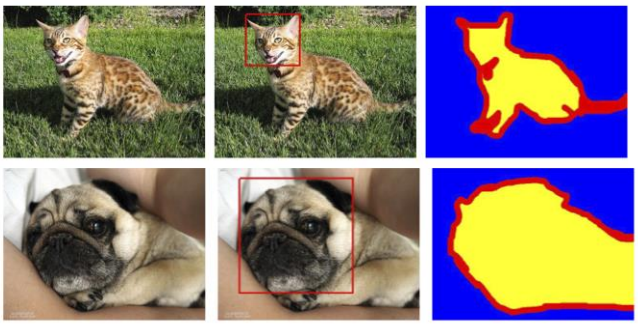
Objective: Build a computer vision application that can utilize the webcam and detect and classify images of dogs and cats from live video feed.

Introduction:

In this project a computer vision application the machine learning library known as tensorflow has been used to detect and classify the images of dogs and cats from the live feed captured by the webcam. This project uses SSD MobileNet V2 model trained on the Oxford-IIIT Pets dataset. The individual parts are integrated together to detect and classify the pet breeds.

Dataset:

The Oxford-IIIT Pets dataset contains 37 classes of pets with each class containing approximately 200 images. The annotations of each images consist of the breed, species name and a bounding box around the head of the animal. It also has a foreground-background segmentation at the pixel level.



Platform:

A detection API was primarily built on the basis of Object Detection Tutorial from GitHub which could process the inference graph and label files from any models and combine with the open camera module so that the objects that the model was trained previously on could be detected. This made possible to import and verify various models. One of such models were ssd_mobilenet_v1_coco_11_06_2017 which was trained on COCO dataset. The API was successful and search for different models was started.

Model:

Initial attempts to build a model from scratch were made but due to tremendous hardware requirements to accomplish such a task, the attempts were dropped. Other alternative ways such as using Google's cloud ML engine were expensive so a pretrained model had been used. However, the procedure to train a model on a dataset and export the inference graph were cleared learned from this project.

Transfer learning was used on the model SSD MobileNet V2 which is pretrained on COCO dataset to further train on our pet classification dataset so that it can classify pet breeds. Other attempts were made to find different models such as YOLO and Keras that were trained on the same dataset, but none were found.

Training and Evaluation:

The dataset was split into three parts training, validation and testing. For each class of breeds, 50 images were used for training, 50 images were used for validation and the rest were used for testing.

There were three tasks defined primarily. They were,

1. Detect the pet in the image.
2. Pet Family Classification (Dog or Cat).
3. Pet Breed Classification (Name of the breed)

The model was evaluated based on three methods: Shape only, Appearance only and Shape and Appearance only.

Shape only: The maximum confidence level of cat face vs maximum confidence level of dog face. Whichever is higher will be used to classify the family.

Appearance only: This method is used to accurately detect the pet's body in order to enhance the accuracy.

Shape and Appearance Only: This uses the summation of shape only and appearance only methods and achieves an accuracy rate of 94.89%.

Method	Accuracy
Shape only	94.21
Appearance only	94.88
Shape and Appearance only	94.89

Limitations:

The following cases observed as failure in the model when an appearance only layout was used.

1. Pup images were incorrectly identified as cats due to their striking similar face shapes.
2. Bengal cats sometime identified as Egyptian Mau.
3. Some furry English Setter was identified as English Cocker Spaniel due to similar appearances.

The problem with Shape Only was that if the faces of the pets were not clear then misclassification was done by our model.

However, these failures were rectified once the shape and appearance only method was implemented where the information of head is used.

Contribution:

We tried to add a new class of dog breed "Dobermann" to this dataset. 40 images of the dog breed were collected from google and tried to train the model. The training process resulted in multiple system crashes due to insufficient resources. The accuracy was calculated using accuracy function from metrics_impl.py which checks the predictions against the labels. Due to substantial difference between the number of images used for training and validation of other classes and our new class, the model started misclassifying and the accuracy was affected.

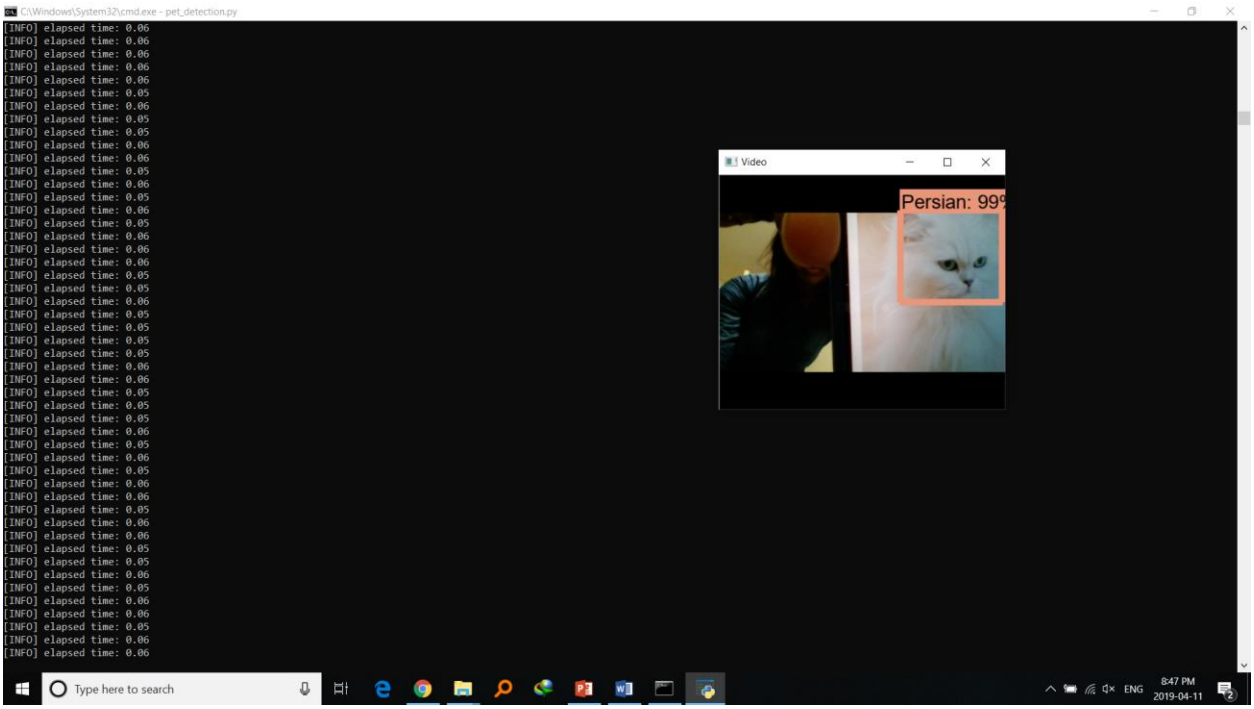
Another mistake we made was presenting the images in a wrong way. The images we presented to models did not have boxes on the faces of the pets nor the foreground-background segmentation of the pet bodies, this had to be done for each and every image. Due to these difficulties, the trained model was scraped off and no further steps were taken towards training a model and focus was diverted on understanding how this model was built.

<u>Multi-threading:</u>

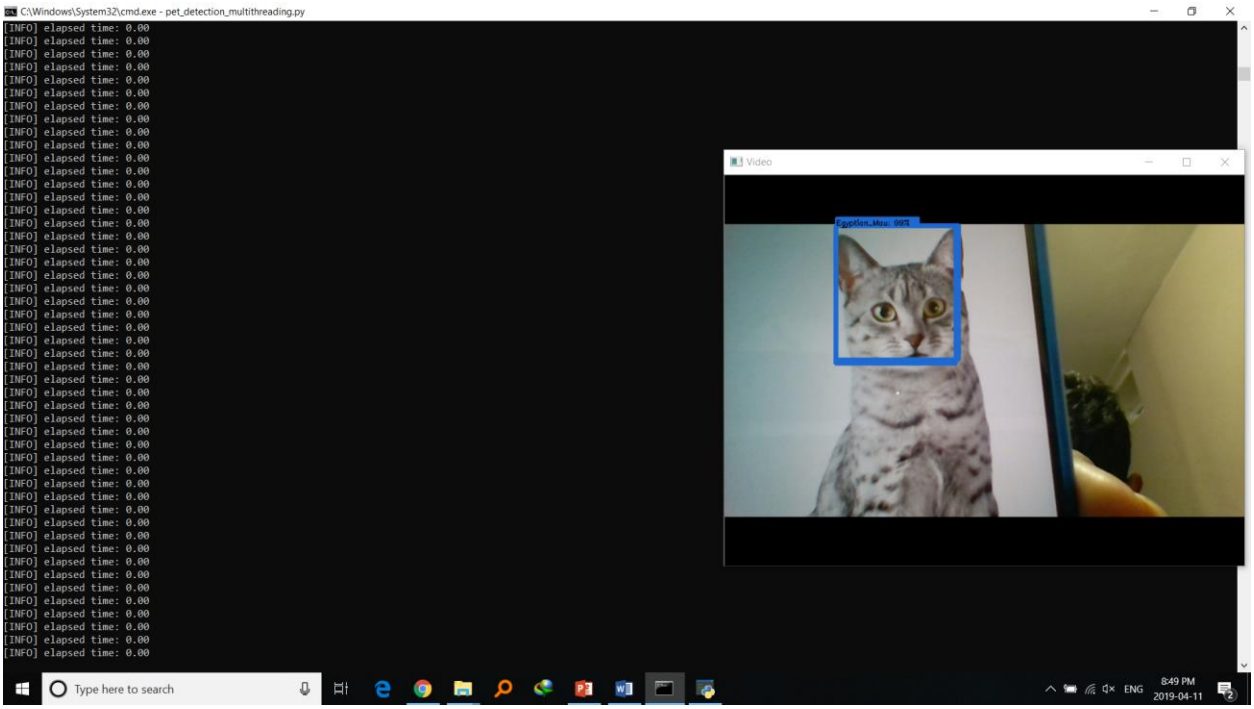
Our concrete contribution to this project would be introduction of multithreading. Multi-threading makes use of the unused CPU resources and boosts the processing speed of this application. Normally the pet detector was working at a frequency of 0.05s and when the multi-threading was utilized the pet detector sped up by 50ms and detection was continuous. This utilization of multi-threading made a difference in the latency of the existing application.

The following images shows the difference in terms of elapsed time.

Without Multi-threading:



With Multi-threading:



Applications:

Potential applications for this API include:

1. Hey Google, Where's my pet? Integration of Google Assistant and home camera system to locate pets.
2. Marketing on social media: Social media could use this API to detect pets from live feed and promote relevant pet products.

Conclusion:

Although this model was relatively smaller than other object detection models that use COCO or Pascal datasets, we were able to get a deep understanding on decision trees and machine learning part by real time implementation. With adequate time and resources, we might be able to build a new model from scratch or improvise this same model by adding new class.

Models considered before finalizing:

Model	**Dataset**	**Reasons for discarding**
faster_rcnn_resnet101_kitti_2018_01_28	KITTI	Technical difficulty (Couldn't find label files)
faster_rcnn_inception_resnet_v2_atrous_oid_2018_01_28	PASCAL	Very Slow
faster_rcnn_inception_resnet_v2_atrous_oid_14_10_2017	PASCAL	Very Slow
SSD MobileNet V2	Oxford-IIIT Pets	NA (Finalized)

References:

1. Dataset from: http://www.robots.ox.ac.uk/~vgg/data/pets/
2. Trained model used from: https://github.com/pillarpond/pet-detector-android