Text

Description automatically generated with medium confidence

2021-CT5103

Case Studies in Data Analytics

Assignment 1

Real-time Object Detection and Classification

Submitted by:

**Rahul Kumar** (MSc. Data Analytics - 20230113 – [r.kumar5@nuigalway.ie](mailto:r.kumar5@nuigalway.ie))

**Mudit Srivastava** (MSc. Data Analytics - 20230113 – [m.srivastava1@nuigalway.ie](mailto:m.srivastava1@nuigalway.ie))

# 1. Pipeline Design

The Computer vision pipeline consists of the following stages:

**INPUT**

* **Video Reader**: The video file is read into the pipeline by creating a **VideoCapture** object using the OpenCV2 library. This object processes the video and returns a stream of individual image frames on which further tasks can be performed.

**MODEL CASCADE**

* **Object Detector:** The individual frames returned by the **VideoCapture** object is passed through a SOTA object detector model – **TinyYOLOv3.** The model returns the detected object in a list with each object’s class ID, confidence score, and several bounding boxes associated with it. We perform a **Non-Max Suppression** on each object bounding boxes and select the one which has the highest confidence score.
* **Attribute Classifier (Car Type):** We retrained the **MobileNet** object classifier for our sedan and SUV classes which was scrapped from the internet using Google and Bing Image Crawler. Once the images were acquired, performed some pre-processing and data augmentation, and lastly used transfer learning approach on the classifier to fine tune it according to our use case.

The bounding boxes from the object detector are then passed to the classifier to find the sedan/SUV class of the car.

**Design Decisions:**

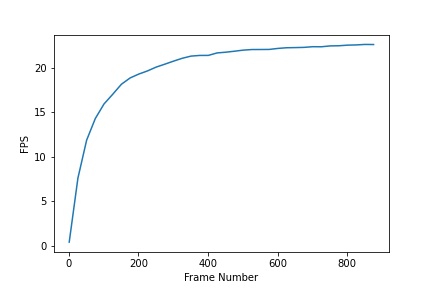
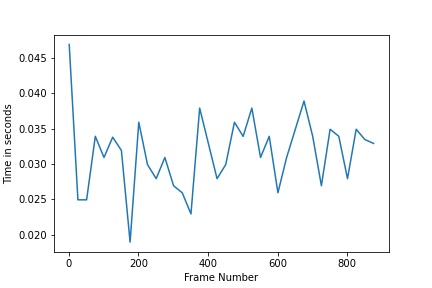
Used **Producer – Consumer approach** in designing the above pipeline where, the **Video Reader** is implemented in a *different thread* which produces the frame and places it in a **queue**. The **model cascade** is*implemented in another thread* which consumes the frame from the queue and perform either only *object detection* (Query-1) or runs the whole model cascade which gives the *car’s attribute* (Query - 2) depending on the choice selected by the user. The producer - consumer model is implemented using **multi-threading** to help the pipeline run more efficiently by running the threads in parallel and enabling the communication between them using a queue which transfers the frames from 1 end to another.

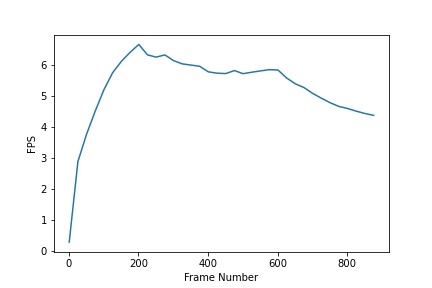
# 2. Pipeline Output

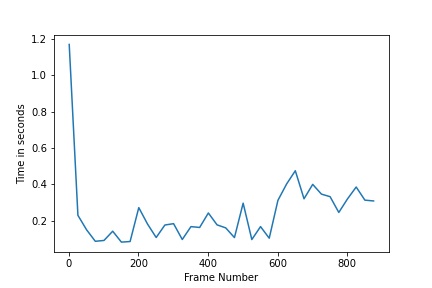
Depending on the Query type the pipeline returns the final video (having the count of cars in each frame and the FPS) along with a csv file to compare the results with the ground truth data.

**Query 1 output:**  The plot on the *left* gives us the **Event Extraction graph** – It is the time taken by **TiniYOLOv3** to process each frame in finding the object and creating the bounding boxes.

The plot on the *right* gives us the **throughput time** - It is the time take from start to finish of the pipeline.

****

**Query 2 output:**  The plot on the *left* gives us the **Event Extraction graph** – It is the time taken by **TiniYOLOv3** **+ MobileNet Classifier in Predicting** the car’s attribute for each frame. The plot on the *right* gives us the **throughput time** - It is the time take from start to finish of the pipeline.



We can see that the query 2 takes larger time in processing each frame: That is due to the reason that we are running both the TinyYOLOv3 object detection, find the bounding boxes, passing it to MobileNet classifier.

**Query 1** takes **~38 seconds** and **Query 2** takes **~200 seconds** of processing time.

# 3. Model Training Evaluation

## 3.1 Data Preparation

To scrap images of SUV and sedan from the web, **icrawler** package have been used. The 2 different web image crawlers from the package are used.

* GoogleImageCrawler
* BingImageCrawler

For the task two iteration of image fetch was done. Firstly, filters were set to only fetch the small images for the classes mentioned and then in second, some medium size images. Then later they were manually mixed to produce a small to medium sized image set, as the size of bounding boxes from the tiny-YOLO were expected to be in this range.

## 3.2 Data Pre-processing

To preprocess the data in the proper input format for the model, **Image Data Generator** from **Keras pre-processing package** is used. It is used with **MobileNet’s preprocess\_input function**.

The data is then split in training set and validation set, with **80%** and **20%** of the data, respectively. These two datasets are created with the help of image generator that we created in the previous step and while assigning, images are also scaled to required fixed target size of (*224,224*) with defined color mode.

## 3.3 Model Architecture

As we have considered **MobileNetV2** as our model for classification task, we defined it from **Keras** applications package and load the predefined **ImageNet weights**. We made sure to not download the top for it i.e., the last layers of the model as we will be attaching our own set as part of transfer learning.

We then made the **base model** *non-trainable* and defined the architecture of the model. On the top of the base model, we attached a *Global average pooling layer*, followed by a *dropout layer* and then a *dense* *layer* with *sigmoid activation*. We kept in mind to have the training parameter as false when passing inputs through the base model as we want it to run in inference mode. As base model accommodate batch normalization layers and we wanted to keep them in inference mode for the time and will unfreeze the model for fine-tuning later.

## 3.4 Model Training

The model is then compiled with *Adam optimizer* for optimization and *binary cross entropy* as loss function, also Binary Accuracy is used as the metric for evaluation of model in each epoch.

Finally, model is trained on the training data and validated on validation data set, in each epoch. As base model is still freeze, therefore, weights for only the layers added as part of transfer learning are modified.

## 3.5 Model Fine-tuning

We now unfreeze the base model but as we defined it with training equals false at the start, it would not be fully trainable but would run in inference mode where weights of layers other than batch normalization will only be modified.

Now we compile the model using same loss function and metrics, but we lower the learning rate for optimization in Adam optimizer as we only want the whole model to tune a bit on whole data and have a nice improvement in training, we do not want to overfit it.

## Results

After the fine tuning, the model was able to achieve the binary accuracy of more than 80% on training data as well as validation data.

# 4. Pipeline Optimisation

We used the **multi-threading** library to implement the producer-consumer model of the pipeline. Creating 2 threads, 1 for *extracting the frames* from the video stream and placing it in *queue* and the other thread to *consume the frame* and process it as per the *model cascade*. This should have increased the *throughput* of the pipeline but there was no significant improvement. This is due to the limitation of python’s **GIL – Global Interpreter Lock** which synchronizes the execution of threads such that only one native thread can execute at a time even if run on a multi-core processor. This limits the pipeline to truly run-in concurrent manner.

One solution is to use **multi-processing** which creates 2 separate processes which can truly run in parallel by bypassing the GIL limitation. But the **Jupyter notebook, windows 10 and python v3.6** have some *coupling issues in implementing multi-processing*. Other solution is to use **JAVA** to implement the pipeline which supports multi-threading much better than python.

Thus, to optimize the efficiency of our pipeline, we focused to *optimize our classifier* as that was expected to enhance the overall performance. Below mentioned are the two optimizations that we did for the MobileNet classifier.

## 4.1 Data Augmentation

Four different types of data augmentation techniques are used with image data generator to produce more variety in data:

• Rotation of images by max of 10 degrees

• Zooming in with a factor of 15%

• Shifting of images height-wise by 0.5 unit

• Horizontal flipping of images

With variety in data, the classifier was expected to learn better and therefore, in-turn predict better.

## 4.2 Callbacks

The callback technique of reduction in Learning rate when the learning stagnates, is used to get the most out of *Adam optimizer*.

**Early stopping callback** is also used to monitor the validation loss as if it does not change much for the period of patience defined, it halts the training. As that is often the point when model starts to overfit.

**Note**: The parameters for both the callbacks are defined just by trial and error, therefore not optimum.

**Without Optimizations:**

**With Optimizations:**

As we can see from the results above (full result in code), the optimized classifier has learned better and provided a better accuracy on validation set than the non-optimized one. Also, we can see, non-optimized classifier overfitted on training data, with accuracy of around 89% on training set and just 81.67% on validation set, same trend with loss. The problem seems solved in the latter case as loss for both data sets is comparable and validation accuracy is bit higher than training accuracy, as expected, due to the use of dropout layer in the model.

When used both the model in our pipeline, the results were:

* **Non-optimized model**: **query 1 - F1** **score** as **84%** and for **query 2 - F1 score** (69% for sedan and 5% for SUV) as **37%**.
* **Optimized model**: **query 1 - F1 score** as **84%** and for **query** **2** **- F1 score** (69% for sedan and 23% for SUV) as **46%**.

# 5. Design Strengths and Weaknesses

## 5.1 Strengths

* The pipeline is made using multi-threading approach in which vide reader produces a stream of frames (producer) placing it in a queue which is then consumed by the mobile cascade. Using Queue for communication between the threads enables us to run them in parallel.
* The classifier is trained on images of various resolutions, therefore, is robust in classification of cars in small bounding boxes and well as big ones.
* As image data generator is used with MobileNet pre-processing, images would be shuffled and converted to proper format as needed by mobileNetV2.
* None of the top layers of MobileNetV2 are used and only base model is considered, therefore able to train the last layers directly on SUV vs Sedan dataset, having the ImageNet weights in the base model. This provided better classification.
* Used dropout regularization layer to combat overfitting.
* The fine-tuning of model with trainable base layers and very less learning rate, provided improvement in accuracy as model got a bit focused on the classification at hand.
* The use of Adam optimizer which has an advantage of using per-parameter learning rate that improves performance on the tasks like computer vision and natural language. And therefore, it provides fast converge as compared to other optimizations like AdaGrad, RMSprop, etc.

## Weaknesses

* Due to python’s GIL (Global Interpreter Lock) issue, the threads do not run in parallel, only 1 thread is active for a given time.
* The bounding boxes created by TinyYOLOv3 is not as accurate as other object detector present. This is because tinyYOLOv3 is trained with a smaller number of layers, resulting in greater speed of processing but with sacrificing the accuracy.
* The bounding boxes created after non-max suppression do not bound the complete object, i.e., is smaller than the object itself. When trying to change the bounding boxes manually using simple array manipulation, results in a smaller number of bounding boxes and is not able to detect as much as before – This is due to the overlapping boxes which is suppressed in non-max suppression stage.
* Even though the MobileNet model is giving good training and testing results, when passing the bounding boxes from the object detector, the accuracy of classifier is suffering. This is because the bounding box is not able to cover the whole object and the MobileNet is losing crucial information which is needed for it to work better.
* Due to limitation of computational power had to tune the hyperparameters for the model and callbacks by trial and error, therefore not optimal.
* As per few articles ADAM may get stuck in local minima and provide poor generalization but provide convergence in fewer epochs. SGD + momentum is recommended as better substitute, as it aims to find global minima, but due to this reason takes much more epochs to converge.

# 6. References

* <https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_gui/py_video_display/py_video_display.html>
* <https://pjreddie.com/darknet/yolo/>
* <https://www.spritle.com/blogs/2019/12/20/real-time-custom-object-detection-using-tiny-yolov3-and-opencv/>
* <https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a>
* <https://www.tensorflow.org/tutorials/images/transfer_learning>
* <https://keras.io/api/applications/mobilenet/>
* <https://stackoverflow.com/questions/54489332/python-multithreading-producer-consumer-pattern>
* <https://stackoverflow.com/questions/21104664/extract-all-bounding-boxes-using-opencv-python>
* <https://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_basic_image_operations_pixel_access_image_load.php>
* <http://anuragjain67.github.io/writing/2016/01/15/problem-with-multithreading-in-python>
* <https://medium.com/python-features/pythons-gil-a-hurdle-to-multithreaded-program-d04ad9c1a63>
* <https://blender.stackexchange.com/questions/8530/how-to-get-python-multiprocessing-module-working-on-windows>
* <https://developers.google.com/search/docs/advanced/crawling/overview-google-crawlers>
* <https://keras.io/api/preprocessing/>
* <https://whatdhack.medium.com/reflections-on-non-maximum-suppression-nms-d2fce148ef0a#:~:text=Non%20Maximum%20Suppression%20(NMS)%20is,out%20of%20many%20overlapping%20entities.&text=Most%20object%20detection%20algorithms%20use,detected%20rectangles%20to%20a%20few>
* <https://paperswithcode.com/method/non-maximum-suppression>
* <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>
* <https://towardsdatascience.com/a-bunch-of-tips-and-tricks-for-training-deep-neural-networks-3ca24c31ddc8>
* <https://stackoverflow.com/questions/43979449/higher-validation-accuracy-than-training-accurracy-using-tensorflow-and-keras>
* <https://prodi.gy/docs/computer-vision>
* <https://stackoverflow.com/questions/15589517/how-to-crop-an-image-in-opencv-using-python>
* <https://stackoverflow.com/questions/21879029/opencv2-codec-support>

# 7. Download Link for Submission Files