Jose Moti

CE17B118

ce17b118@smail.iitm.ac.in

CS6886: Systems Engineering for DL

ASSIGNMENT 1

PART 1

RESNET50

The pretrained model is downloaded directly from torchvision

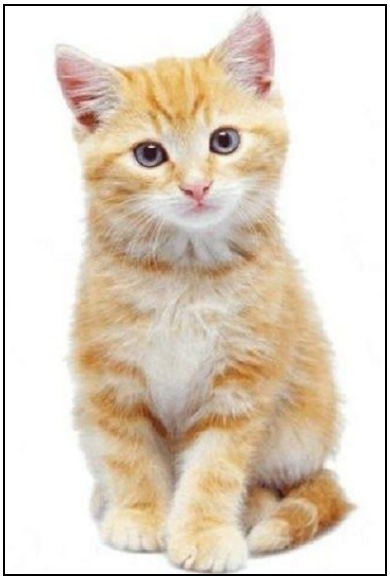
```
torchvision.models.resnet50(pretrained=False, progress=True, **kwargs)
```

The model architecture is downloaded along with it. The onnx model was exported from the torchvision model.

Going to the folder there is :

- **imagenet_labels** : a dictionary for predicting the object in each given input image
- **resnet50.onnx** : which is the exported model itself

The onnx model was successfully verified and both the models were tested with the same input image just to verify both gave the same output.



The top 5 answers for the image as given by the model are:

283: 'Persian cat',

281: 'tabby, tabby cat',

282: 'tiger cat',

285: 'Egyptian cat',

728: 'plastic bag'

* The outputs show that the pretrained model works well for this given image.

SSD_RESNET34

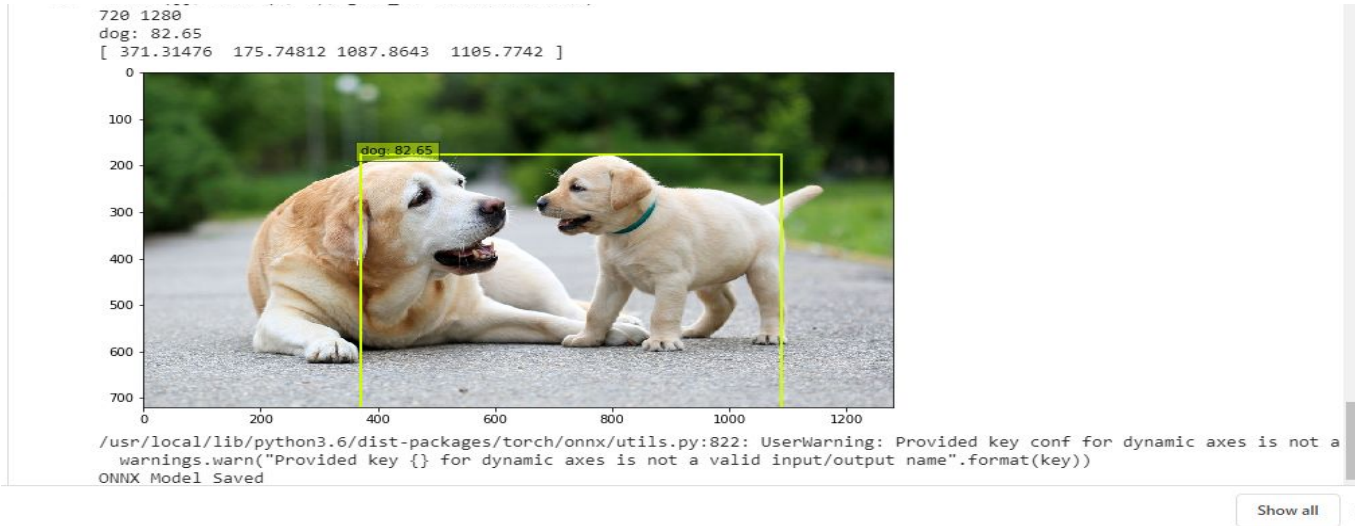
The pretrained model is downloaded from
https://zenodo.org/record/3236545/files/resnet34-ssd1200.pytorch?download=1

And the model is obtained from
https://github.com/itayhubara/MLPerf-SSD-R34.pytorch/tree/master/ssd_r34_2m

Going to the folder there is :

- **ssd_r34.py and base_model_r34.py** : which both contain the entire model
- **plot_detections.py** : to reconstruct the image with bounding boxes.
- **coco_labels** : a dictionary for predicting the object in each bounding boxes
- **resnet34-ssd1200_20.pt** : contains the pretrained weights.
- **ssd_resnet34.onnx** : which is the exported model itself

Both the PyTorch model as well as the onnx model were tested on the given image and both were found to give the same outputs. This can further verify the fact that export was done correctly. The output of the model after processing in order to generate the bounding boxes and plotting it accordingly is given below. Screenshot taken from colab notebook.



* One major difficulty faced was exporting the full model with post-processing is done was not possible as onnx was unable to identify the functions. So I had to do the post-processing after it was converted to onnx model.

MOBILENET

The pretrained model is downloaded from the given link:

https://drive.google.com/open?id=1W61GiwsJy4HvIWnxw_WlEmhjftqLy4M_

The full code inspired by

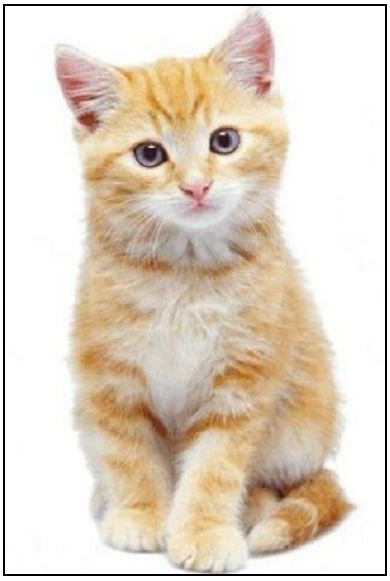
https://github.com/ruotianluo/pytorch-mobilenet-from-tf

The model was successfully exported to onnx format.

Going to the folder there is :

- **imagenet_labels** : a dictionary for predicting the object in each given input image
- **mobilenet.onnx** : which is the exported model itself

The onnx model was successfully verified and both the models were tested with the same input image just to verify both gave the same output.



The top 5 answers for the image as given by the model is:

```
282: 'tiger cat',

283: 'Persian cat',

281: 'tabby, tabby cat',

285: 'Egyptian cat',

287: 'lynx, catamount',
```

* The outputs show that the pretrained model works well for this given image.

* It was difficult to define the whole model here as the model architecture has given and the pre-trained dictionary had different parameter names.

* By comparing with Resnet 50 predictions we can see that they give different predictions but all the top 5 predictions are based on different species of cat.

SSD_MOBILENET

The pretrained model is downloaded from the given link:

https://storage.googleapis.com/models-hao/mobilenet-v1-ssd-mp-0_675.pth

The architecture and the codes were taken from this link:
https://github.com/qfgaohao/pytorch-ssd/blob/master/README.md

Going to the folder there is :

- **mobilenet-v1-ssd-mp-0_675.pth** : a dictionary containing pretrained weights
- **vision** : It is a directory that contains the architecture-related things of the model.
- **ssd_mobilenet.onnx** : which is the exported model itself

The onnx model was successfully verified and both the models were tested with the same input image just to verify both gave the same output.

Another main thing to note was the dataset used in the training of SSD_Mobilenet. It was a COCO dataset but only had 21 classes including background compared to the 81 of the original COCO dataset. Also, the export to ONNX was not possible with GPU in Colab. It was only possible with the CPU (by turning off the accelerator in Colab.

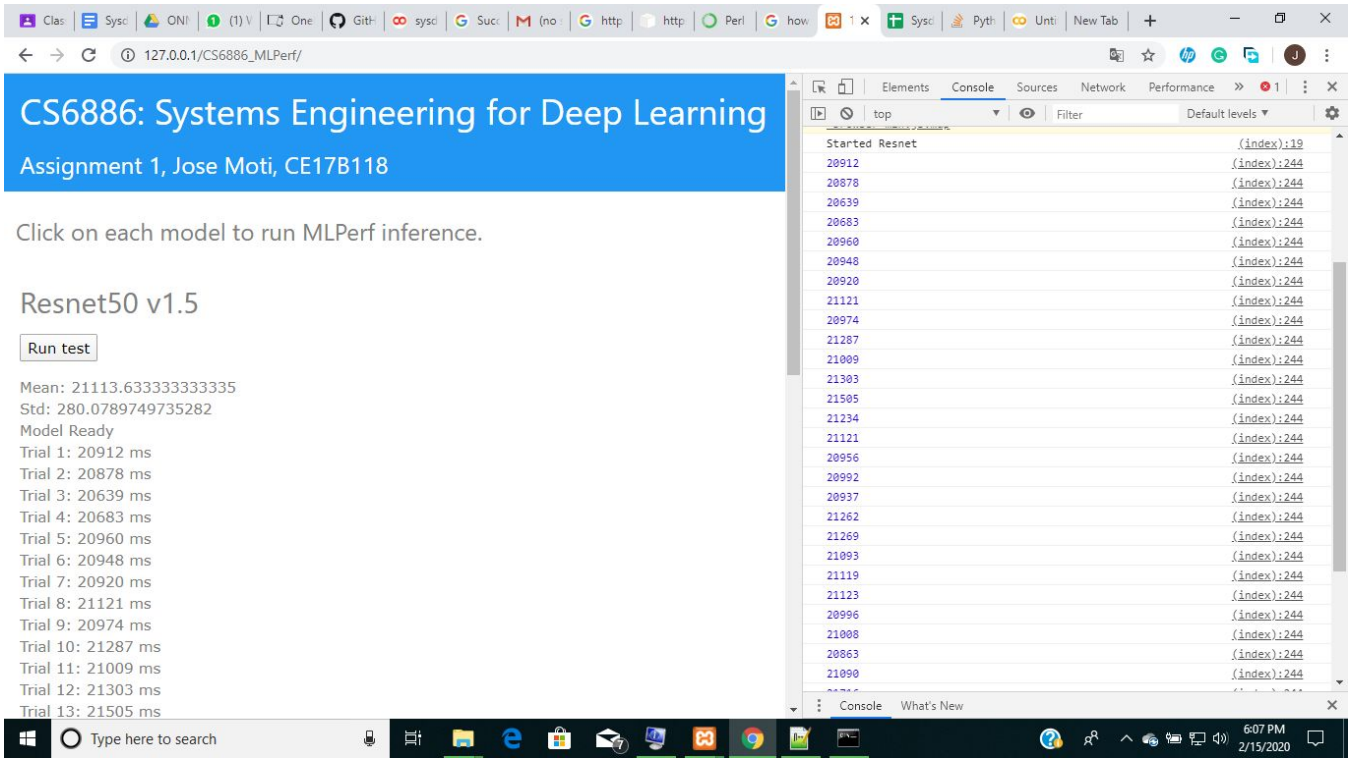
PART 2

ONNX.js

For running in ONNX.js I have borrowed the codes from my friend 'Sooryakiran P (ME17B174)' as I don't have much knowledge in JAVASCRIPT. I modified the codes so that I would be able to host my ONNX files.

The same image is given for all models for the inference session.

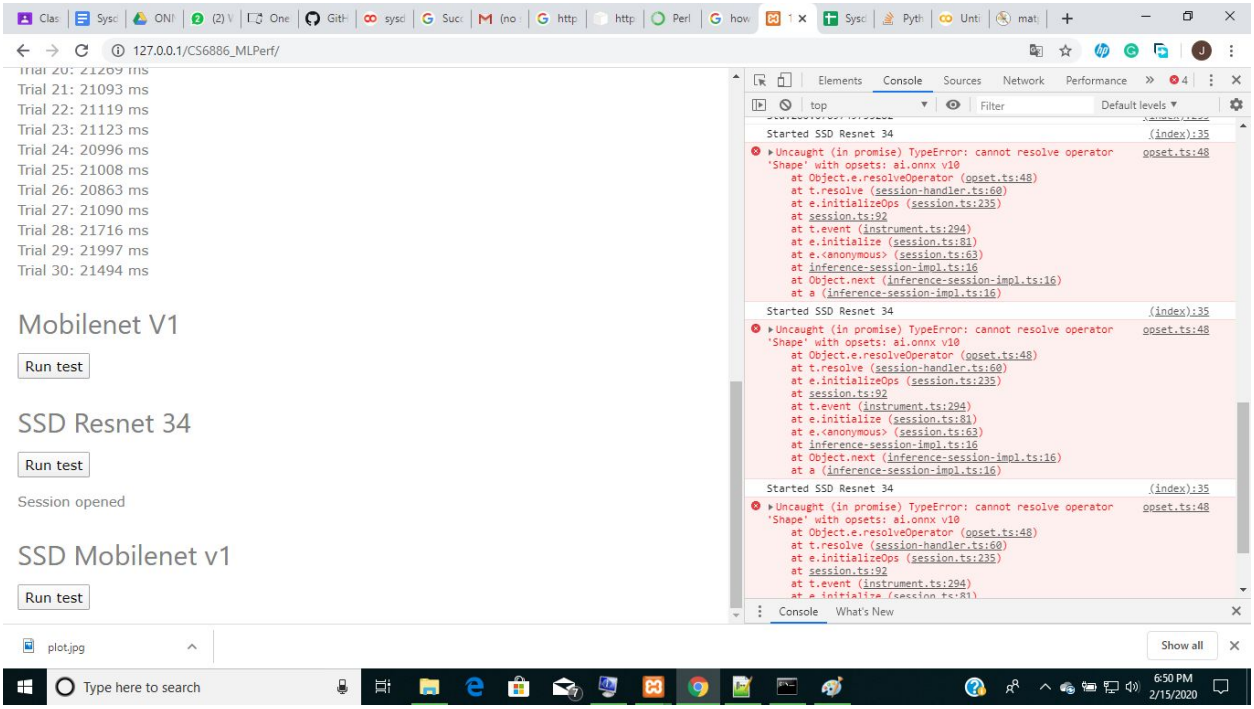
The codes required are given in js_folder.



- The screenshot of the console window while running the resnet50 model is given here. Looking closely we can see that each inference session takes as much as 21 seconds which is too much for a laptop. This is because my laptop always slows down when large files are put into its memory.
- So I ran the model on 4 other laptops from my friends and a mobile phone too. The specs of each of these devices are given below in the table.
- I was not able to infer from the SSD_resnet34 onnx model. There was this following Typerror coming up. ('Cannot resolve operator shape with opsets: a1.onnx v10)
- I tried with my friends' onnx models too but all of them were giving the same Errors.

SSD_Resnet 34 error

The screenshot of the error in the console window is given below.



The reason for this may be some exports in onnx that may not be compatible with js.

I ran the javascript using **XAMPP** which is a local server for hosting from Windows 10. The js_folder which contains all the requires js files were placed inside its source location.

PART 3

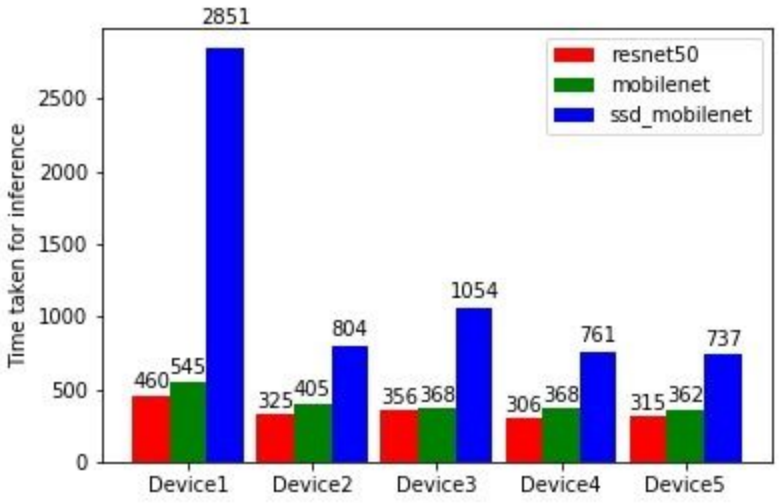
Variations in inferences with devices

30 inferences are done in each device and its mean and standard deviation are recorded as shown in the table below. (SSD Resnet34 was not working)

	Device1		Device2		Device3		Device4		Device 5	
Resnet50	460.6	11.27	325.36	23.2	356.5	37.58	306.86	35.35	315.3	29
Mobilenet	545	4.2	405.26	29.49	368.6	12.1	368.37	28.05	362.3	19.71
SSD_resnet34										
SSD_Mobilenet	2851	46.47	804.47	5.78	1054.2	9.87	761.77	11.8	737.76	12.57

The plots for the 3 networks on 5 devices are given below. Plotting code in **plotfinal.py.** Code also is given in colab notebook.

The value above bar charts corresponds to the average time taken for each inference session.



The specs of each of the devices used are given below.

Device1	Device 2	Device 3	Device 4	Device 5
CPU: Qualcomm Snapdragon 845	CPU: Intel® Core™ i7-7500U	CPU: Intel(R) Core(TM) i7-7500U	CPU: Intel(R) Core(TM) i7-7500U	CPU: Intel(R) Core(TM) i7-9750H
8 Cores	2 cores	4 Cores	4 cores	6 Cores
4 cores @ 2.8 GHz 4 cores @ 1.7 Ghz	2.7 GHz	2.70GHz	1.80GHz	2.60GHz
Memory: 8 GB	Memory: 16GB	Memory: 8 GB	Memory: 8 GB	Memory: 16 GB
Cache: L1 512kB x2	Cache:4 MB	Cache: L1 4096 KB x4	Cache:8 MB Intel® Smart Cache	Cache: 12 MB Intel® Smart Cache
Accelerator: Not used	Accelerator: Not used	Accelerator: Not used	Accelerator: Not used	Accelerator: Not used

Plot Inferences:

- The mean inference time for Resnet50 and Mobilenet almost the same for all devices.
- The mean inference time for SSD_Mobilenet is higher for the smartphone compared to the rest of the devices which are laptops. The reason for this may be the difference in architectures for laptops and smartphones. Also, there is a considerable difference regarding cache from the other devices.
- Device 6 has 6 cores of 2.60GHz while device 1 has 8 cores out of which 4 are just 1.7 GHz. Higher the number of cores, more the parallelization can be done resulting in lower inference times.
- There were accelerators in all the devices but none of them were used while inference.
- The best performance from the plot can be given to device 5. It was actually a gaming laptop and it has 6 fast cores as well as a RAM of 16 GB leading to its best performance.
- Using GPU backend can give much better results since a lot of work can be parallelized.

Pearson Correlation Coefficients

Pearson's correlation coefficient is the covariance of the two variables divided by the product of their standard deviations. It is done here to check how the inference time changes with changes in the model. Since I had only 3 models, I have done it for 3 models only.

	1	2	3	4	5
1	--	0.992178	0.999873	0.995602	0.997512
2	0.992178	--	0.990058	0.999509	0.998509
3	0.999873	0.990058	--	0.993980	0.996259
4	0.995602	0.999509	0.993980	--	0.999729
5	0.997512	0.998509	0.996259	0.999729	--

- The Pearson Correlation plot was drawn in **Google sheets with CORREL(data_x,data_y)** function.
- From the plot, we can infer that the maximum correlation is for **Devices 1 and 3**
- The minimum correlation is for Devices 2 and 3.
- Cannot justify this finding of max correlation as one is a laptop and the other is a smartphone.

Observations and Conclusions

- The specs of a device affect the inference time of an image in a model which is clearly visible in plots given.
- Even a slight increase in 100ms can be a great problem because the dataset in which the inferences are done is usually very very large.
- If we take an ssd mobilenet, for example, the best inference time for an image is 737 seconds. Even this is a bad value when it comes to real-time image processing such as the ones used in autonomous cars. The device would have to infer on about 10-20 frames per second in order to be effective.
- The above case shows why faster and faster processors are required in this Deep Learning era.
- Also, I learnt how to export a PyTorch model to onnx and run this onnx model in the web browser using java Script.

References

- Pytorch documentation
- Github MLPERF repository, pytorch_ssd repository
- Python, Javascript
- Stackoverflow.com
- MLPERF paper
- XAMPP documentation
- Wikipedia

Codes

All the codes were written in Google colab with files being called from my Google drive. I have downloaded the complete folder from google drive along with the .ipynb notebook.

See **sysdl_assignment1.ipynb** for more details.

The **js_folder** contains all the necessary codes and onnx models required.