Developing a Brand Reconition Application Using Deep Learning

Anthony, Haram and Omari

Abstract—As the mobile device has transitioned to becoming the most popular computer in the world, the need for more applications to be developed for the mobile device has increased in demand. We attempt to integrate a deep learning model developed using NVIDEA GPU, PyTorch, and Cuda into a Flutter mobile application. We discussion the process of developing the application as well as any obstacles encountered in the process.

Index Terms—Deep learning, GPU, TensorFlow, PyTorch, Flutter, NVDIA, Colab, Spyder

1 Introduction

Deep learning is the most scalable learning algorithm when compared to older learning algorithms, according to data scientist [1]. Andrew Ng stated, "The analogy to deep learning is that the rocket engine is the deep learning models, and the fuel is the huge amounts of data we can feed to these algorithms." The available fuel for these algorithms could be obtained by the mobile device and the many activities that a user may partake in daily. In 2013 the mobile device surpassed the desktop device to become the most popular computer [2]. The intersection of deep learning algorithms and mobile devices is a unique area of overlapping interest when considering the potential emense amount of data to be accessed through the mobile device and the number crunching capabilities that the deep learning algorithms possess.

Developing a deep learning mobile application requires a specific approach. A strong consideration must be made for the amount and type of computation that the device has available. Deep learning demands a huge amount of resources from the machine during development which requires more high-end machinery compared to traditional machine learning algorithms. Though the equimpment cost and resources needed may be a disadvantage, the benefit is returned through the deep learning algorithms ability to learn high-level features from data in an incremental manner [1]. The figure below illustrates the elimination of the need for feature extracting domain experts during the learning process [3]. To employ the power of deep

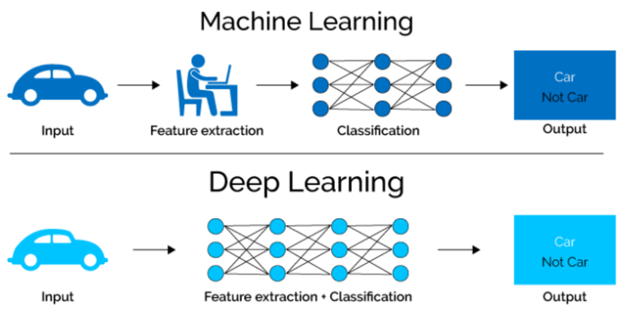


Figure 1: Traditional machine learning contrasted with deep learning.

learning models with a mobile device careful consideration must be given to the size of the network and the demand that it may put on hardware to compute. A reasonable approach is to find a neural network model that does not have a ton of layers thereby reducing the demand on the device [4]. Also, a deep learning model can be flattened to a light version of the model for certain platforms. We will discuss our process and design for obtaining an appropriate model for mobile application.

2 Proposed Design

The initial stages of designing the overall application are critical in being able to choose appropriate technologies as well as neural networks. The design of the brand recognition mobile application is centered around three main components which are 1) camera video capture 2) video data streaming through an api 3) video data being processed by a DNN. Figure 2 illustrates these steps.

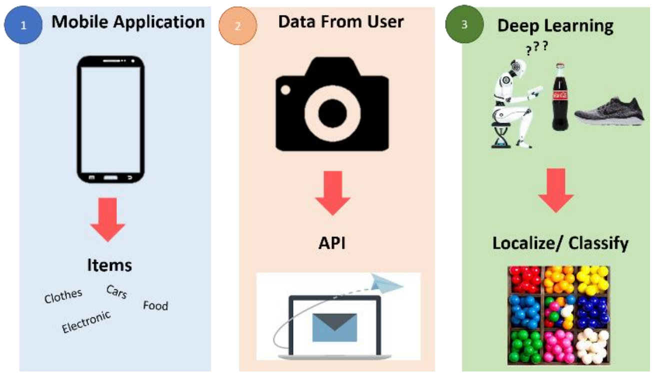


Figure 2: Design Components

For this application the mobile device will be the limiting factor for which potential compromise must be made. Therefore, the first design goal will be to ensure that the accessibility of the application will remain as inclusive as possible. For example, Apple has many generations of the popular iPhone mobile device where each generation of the phone has different specifications and ultimately different computing abilities. It is the goal for this application to be compatible with as many of the iPhone generations as possible. The same logic is also applied across the board for other manufactures as well (Samsung Galaxy, OnePlus, Huawei, Google Pixel, etc). The second specific goal for this deep learning application is to train a network to recognize different classes of sodas. The model will be trained on classes of Coke, Dr. Pepper, and Pepsi. The recognition should also localize the objects within the video capture with bounding boxes. Localization and Classification with a certain amount of statistical confidence make up the entire object detection model. Figure 3 illustrations the three classes of sodas to be detected by the brand recognition model which are classified and bound.

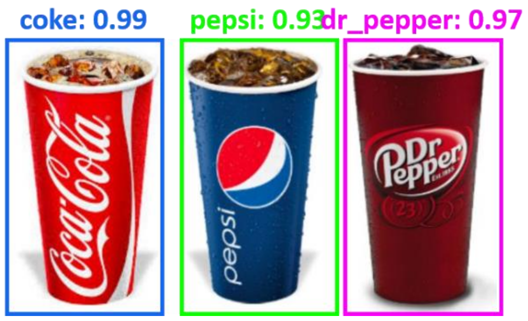


Figure 3: Localized and Classified sodas

The mobile application will function by streaming the video captured data through the API to the DNN model. The model is responsible for classifying and localizing the objects within the video feed. Once the objects are properly detected by the network, they can then be sent back to the viewing window of the mobile device. Figure 4 shows the direction of the data between the API and the camera.

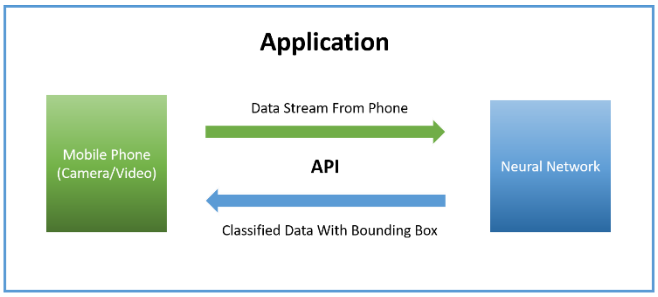

Application
Mobile Phone (Camera/Video)
Data Stream From Phone
API
Classified Data With Bounding Box
Neural Network

Figure 4: Mobile application functionality.

The proposed GPU stack to develop the application consisted of the NVIDIA graphics card, Cuda, PyTorch, and Spyder. The mobile environment is proposed to be developed using Flutter which is a Google UI toolkit [5]. The figure below illustrates the relationship between the proposed tech stack and the mobile environment.

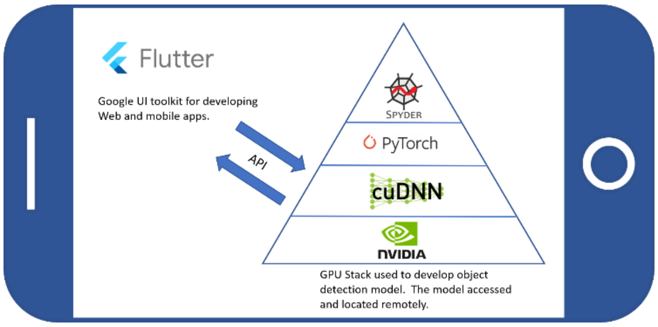

Flutter
Google UI toolkit for developing Web and mobile apps.
API
SPYDER
PyTorch
cuDNN
NVIDIA
GPU Stack used to develop object detection model. The model accessed and located remotely.

Figure 5: Proposed tech stack of NVIDIA GPU, Cuda libraries, PyTorch, Spyder and Flutter application development environment.

While attempting to implement the proposed design there were issues incorporating the proposed tech stack into the Flutter environment. The details will be disclosed in the implementation section of the paper.

3 Implementations

Initially the project was split into two separate projects where the first project contributor (Aram) was slated for developing a Flutter application that detects cats and dogs. The second application was slated for the contributors (Anthony and Omari) to develop a model that could detect different brands of sodas. Due to overlapping interest between the two projects the contributors of both projects decided to merge into one project which would take the Flutter mobile environment from the first project and use it as an environment to which the brand recognition model would be deployed. The members of the newly formed group strategically planned how the technology was to be transitioned over from each project how each component would interface with the newly designed brand recognition mobile application described in the design section of the paper. The specific details that describe the merging and proper interfacing of the technologies from the two projects would prove to be the most challenging aspect of the process. The Figure below illustrates the main features of the two distinct projects before and what would be after the merge.

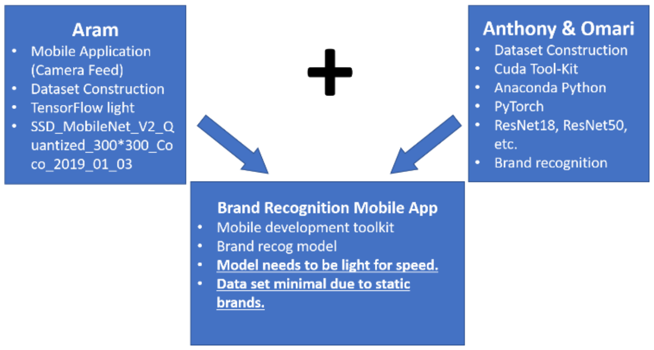
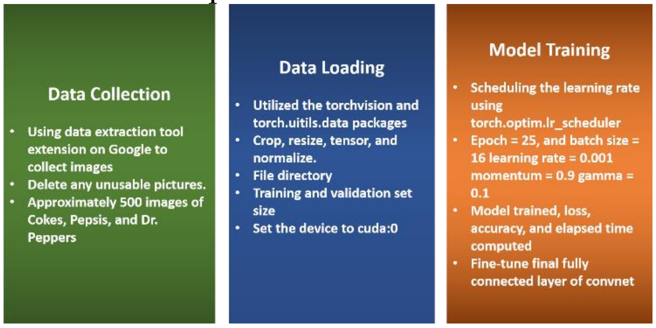

Aram
Mobile Application (Camera Feed)
Dataset Construction
TensorFlow light
SSD_MobileNet_V2_Quantized_300*300_Coco_2019_01_03
+
Anthony & Omari
Dataset Construction
Cuda Tool-Kit
Anaconda Python
PyTorch
ResNet18, ResNet50, etc.
Brand recognition
Brand Recognition Mobile App
Mobile development toolkit
Brand recog model
Model needs to be light for speed.
Data set minimal due to static brands.

Figure 6: Two initial projects merged into one project.

While merging the projects, there were two main obstacles that were encountered. The first was because one project was using PyTorch libraries for all the tensor calculations and technical work where the other project was using TensorFlow. The second obstacle was due to how the NVDIA GPU resources were being accessed. For one project the Cuda library was used to access the GPU resources where the second project Colab was used to access GPU resources. Due to not being able resolve the issue of troubleshoot these problems in a timely fashsion, this paper will report on and demonstrate the development of each individual project.

The first project implementation that will be discussed is the brand regonition application. There were two different models that were experimented with during training and development and both will be discussed in this section. The brand recognition demo was developed using a stepwise process that consisted of collecting data, processing the data, and training on the data using a pre-trained network. Figure 7 illustrates the broad steps used to develop the soda classification model.



Data Collection

- Using data extraction tool extension on Google to collect images
- Delete any unusable pictures.
- Approximately 500 images of Cokes, Pepsis, and Dr. Peppers

Data Loading

- Utilized the torchvision and torch.utils.data packages
- Crop, resize, tensor, and normalize.
- File directory
- Training and validation set size
- Set the device to cuda:0

Model Training

- Scheduling the learning rate using torch.optim.lr_scheduler
- Epoch = 25, and batch size = 16 learning rate = 0.001 momentum = 0.9 gamma = 0.1
- Model trained, loss, accuracy, and elapsed time computed
- Fine-tune final fully connected layer of convnet

Figure 7: 3 stage model development

The images were collected from Google images using a basic search for Dr. Pepper, Coke, and Pepsi then extracted using an extension that allowed for a specied number of images to be downloaded in bulk. After downloading, the data had to be cleaned by removing the non-pertinant images from the collection of images. The images are now able to be loaded. Loading is a processing step that includes cropping, resizing, turning into tensor, and normalizing as the first steps. The torch.utils.data packages from PyTorch are used to make these adjustments to the images. After that stage, the file directories for the images are created by class and the images are separated into training and validation sets. The data is now ready for training and the parameters on set accordingly. The parameters that are set and documented are the learning rate, epoch, batch size, momentum, and gamma [6].

After the transfer learning is completed using the ResNet18 pretrained model, we were able to view some of the predictions [7]. Most of the predictions were accurate but there were some bad guesses by the model. The figure 8 below illustrates two cases where the model predicted Pepsi for an actual Coke and predicted Coke for an actual Dr. Pepper.

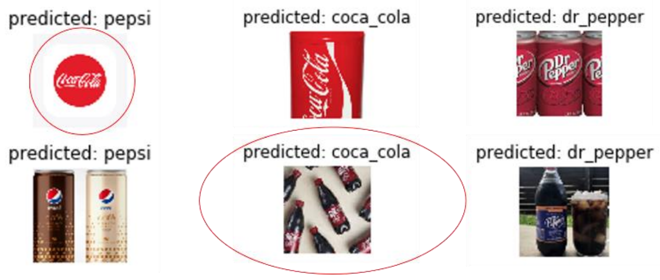


Figure 8: ResNet18 predictions for data.

The second model that was tested by the first group was the Faster rCNN ResNet50 FPN (PyTorch Model Zoo) which is a pretrained network on COCO train2017 and evaluated on val2017. This network can recognize up to 80 object categories. When using this network to evaluate video footage it was seen that the objects in the videos were classified and localized. Detecto was incorporated into the code for inferencing. Detecto is a Python package that is built on top of PyTorch, which simplifies the process of transferring models between two libraries [8]. The goal for this portion of the project was to train the model to recognize sodas based on what was conducted using the ResNet18 model.

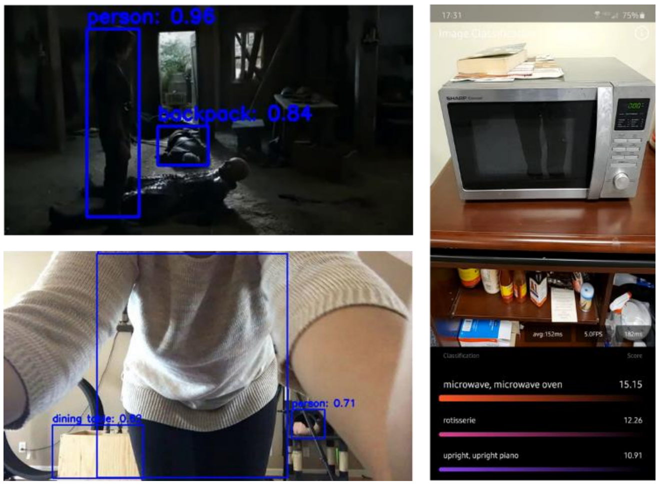


Figure 9: Detecto output top and bottom left. Android application right.

The two images on the top and bottom left of figure 9 are the output from the model developed using Detecto. We were not able to successfully transfer brand recognition to this model, but we were able to run inferencing on some videos and view/save the output from the model in .avi file format. The image on the right in figure 9 is a mobile application developed using Android Studio compiled using Gradle. Within Android Studio, the PyTorch libraries were accessed using import torch. Android studio was used instead of Flutter to launch mobile version just to prove part of the concept that the final product will eventually be a mobile application.

The second application (Aram) was developed using a tech stack that is more like the proposed design for the merged project. This application was developed to detect the difference between cats and dogs. The technology that was used was NVIDIA GPU, Colab, and TensorFlow [9]. The mobile application environment was developed using Flutter. The figure below illustrates the different technologies used to make the application.

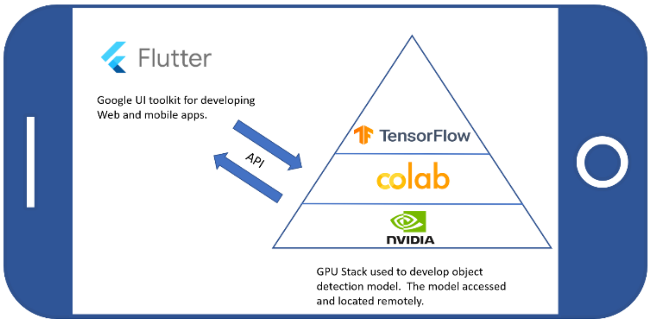

Flutter
Google UI toolkit for developing Web and mobile apps.
API
TensorFlow
colab
NVIDIA
GPU Stack used to develop object detection model. The model accessed and located remotely.

Figure 10: GPU stack for dog and cat detection

The data for this application was downloaded from Kaggle instead of using searches from Google. The bounding boxes were developed using LabellMg. The data was also sorted into a 70/30 percent split between training and testing sets. The labels were saved in XML then converted to CSV with a python script. The TensorFlow record files for training and testing data are also created using a python script. TensorFlow record is a format that is readable by TensorFlow. The next step was to make the label map.pbtxt file that contained the names of the classes to be detected. The model is now ready to be trained on the data. The pre-trained SSD_MobileNet_V2_Quantized_300*300_COCO_2019 was used to train the model where training the model took approximately 12000 steps using the data set. Once the model was trained it was then exported into a TensorFlow-lite model. At this time, the model was ready to be deployed into the mobile application. The figure below is the flow of events starting from training to deploying.

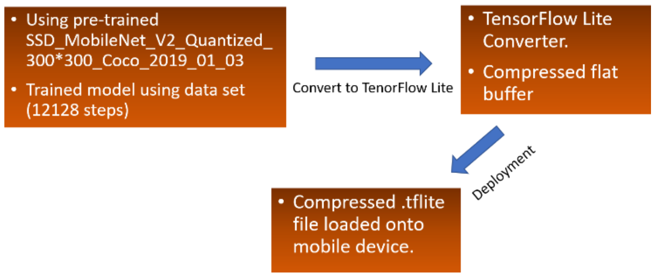

Using pre-trained SSD_MobileNet_V2_Quantized_300*300_Coco_2019_01_03
Trained model using data set (12128 steps)
Convert to TenorFlow Lite
TensorFlow Lite Converter.
Compressed flat buffer
Deployment
Compressed .tflite file loaded onto mobile device.

Figure 11: The three main stages to get to deploying the model on Flutter.

4 Discussion

The paper discusses the development of an object detecting mobile application. There were two distinct development processes that were discussed. The first used a specified GPU stack of NVIDIA, Cuda, PyTorch, and Spyder and experimented with launching the model in a mobile android environment. The second process used a GPU stack of NVIDIA, Colab, and TensorFlow then launched the application in a Flutter environment. The two projects were unsuccessful in the venture of merging the two applications into one application due to the errors that were encountered from transitioning from TensorFlow to Pytorch and working with Cuda versus Colab. Development will continue to take place during the summer to develop the brand recognition model that uses the GPU stack of NVIDIA, Cuda, and PyTorch and incorporated into the Flutter mobile environment.

References

[1] S. Mahapatra, https://towardsdatascience.com/why-deep-learning-is-needed-over-traditional-machine-learning-1b6a99177063, 2018.

[2] O. Andrew, https://blog.islonline.com-/2015/04/16/remote-support-for-mobile-devices-the-simplest-and-most-productive/, 2018

[3] I. GoodFellow et al., Deep Learning, 2017

[4] https://pytorch.org/mobile/home/

[5] https://flutter.dev/

[6] C. Fotache, https://towardsdatascience.com/object-detection-and-tracking-in-pytorch-b3cf1a696a98 , 2018

[7] https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

[8] https://detecto.readthedocs.io/en/latest/index.html

[9] https://www.tensorflow.org/

[10] https://towardsdatascience.com/training-yolo-for-object-detection-in-pytorch-with-your-custom-dataset-the-simple-way-1aa6f56cf7d9

[11] C. Peng, A Large Mini-Batch Object Detector